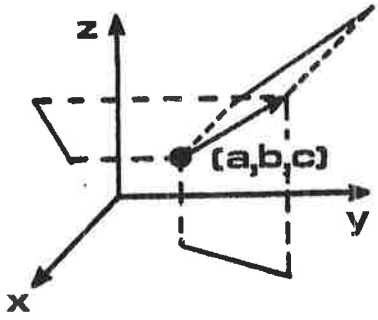
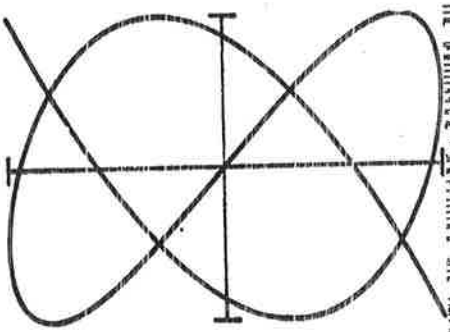
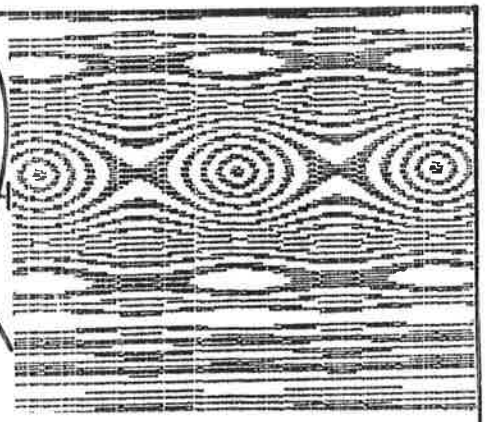
APPLIED SINCLAIR

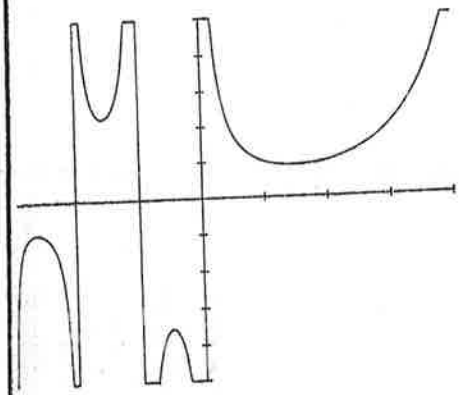
Subroutines and Programs for the Mathematically Minded

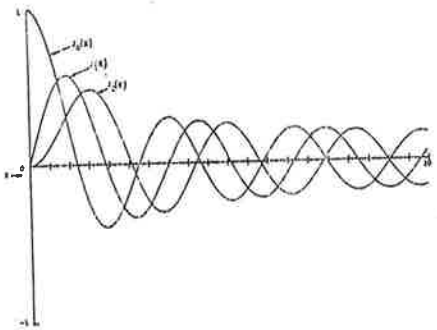
Richard Booth

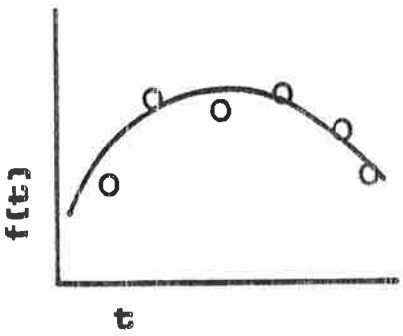












$$\int_0^{\infty} t^{u-1} e^{-t} dt$$

$$\begin{bmatrix} 1 & 2 & \cdots & N-1 & N \\ 0 & 1 & & N-2 & N-1 \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & \cdots & 0 & 1 \end{bmatrix}$$

$$\frac{d^2Y}{dx^2} = -Y$$

$3.1415926535_{10} = 11.001001000011111101101010100010001_2$

Applied Sinclair
Subroutines and Programs for the Mathematically Minded

BLANK PAGES IN ORIGINAL NOT REPRODUCED

CONTENTS

INTRODUCTION

Let me ask the question most frequently asked of owners of home computers: "What on earth do you do with it?" Recently, my answer has become: "I use it as a learning toy." This book is based on the premise that the home computer is best used as a learning device; and if you're not careful, you'll find yourself spending hours with it, learning again all of those things that you had forgotten from school years past. The reader and user of this book will experiment with programs and subroutines that integrate functions, invert matrices, find roots of functions, plot curves and contours of surfaces, and solve differential equations. He may find renewed interest and greater understanding of the mathematics behind some of the widely-used computer algorithms. He may learn some new programming techniques, since many demonstration programs are followed by suggestions for altering the main program and/or subroutines to produce customized results. And he may even find some genuine uses for some of these programs and subroutines, (as have I).

The book is based upon 25 programs and subroutines, written in BASIC for the Sinclair ZX81, and TS2068 computers. The programs and subroutines are presented in 59 demonstration problems. No mathematical fluency or ability is required on the part of the reader to try out any of the demonstration programs: each is presented in a step-by-step table of operations.

Many of the central programs and subroutines call upon other subroutines within the central set of 25 programs and subroutines. In addition, the main programs of each demonstration problem consist mainly of subroutine calls to central subroutines. In this manner, much repetition has been spared. For example, ROOTN, the root-finding program found in chapter 6 calls MXINV, the matrix-inversion subroutine found in chapter 5.

The central programs and subroutines have compatible line-numbers, so that they all may be loaded into memory at the same time. The first subroutine, VINAX, a subroutine to find the minimum and maximum of a data-vector, starts at line-number 1000. All of the demonstration main programs are found between line-numbers 1 and 999. The central programs and subroutines are numbered 2 apart, so that modifications to programs may be easily made.

I have attempted to make the programs and subroutines as general as possible. For example, the linear least-squares regression subroutine, REGRP, found in chapter 6, is designed for any general sum of fitting functions, (powers of x, exponentials, etc.). Linear regression is a sub-category of these procedures and it is presented as a demonstration program only.

The book is organized as follows: Each chapter starts with a short and corny introduction just to start things off. Next, the program/subroutine listings are given, followed by a "spec-sheet" giving the necessary inputs to the subroutine, an example call, and

a description of the algorithm, or a line-by-line description of the subroutine or program. Several problems are then presented in this form: A statement of the problem, a listing of a main program to enter, a table of subroutines and programs to enter, a step-by-step set of instructions to use the demonstration program, a discussion section to give the user some feedback as to whether his main program is working or not, and finally a suggestion section to give the user some motivation to modify the programs or subroutines, or invent new algorithms.

The last step in each of the demonstration programs is the theme of the book: to get the reader/user involved in programming and problem-solving by modifying the programs to his/her need or desire. There are a large number of infinitely more sophisticated programs which run faster or more accurately for specific problems such as those in the demonstration programs. However, the programs and subroutines in this book are a good collection for general purposes. The reader who needs a faster or a more accurate approach should turn, (quickly), to the reference section. In addition, I believe there is enough documentation with each program and subroutine to enable the serious user to modify the routines to more specific purposes. For example, the general Simpson's Rule Integration subroutine, INTEG.1, found in chapter 4, might be modified so that the subroutine returns with an error if the computed estimated error gets larger than a given amount.

Several appendices are included at the end of the book: One-line functions, Physical and Mathematical constants, Units conversions, and a table of central programs and subroutines.

I own a Sinclair ZX80 computer, upgraded with an 8K ROM, and 16K RAM. I also own a TS2068. I acquired both of these before TIMEX dropped out of the home-computer market, but regret neither purchase. The ZX80 doesn't have a "SLOW" mode for flicker-free graphics. Consequently, none of the programs in this book make use of the "SLOW" mode found on the ZX81. In addition, the changes needed to get the programs to run on the TS2068 are minor ones. The color-graphics and other enhancements available on the TS2068 are not used here, in an effort to keep the programs and subroutines as compatible as possible between the ZX81 and TS2068.

It is possible to load into 16K of memory all of the programs and subroutines in this book, with enough memory left over to write a main program. This is the most time saving method of using the book: each demonstration problem requires the user to enter at least one of the programs or subroutines, and if they are already in memory, that much time has been spared. In addition, any of the other routines may be used in modified main programs.

The Sinclair computers offer a superior BASIC and monitor, in my opinion; if not in speed, then in style. It is possible to dimension an array by a variable: DIM X(N) is acceptible. This book takes advantage of the VAL function, which evaluates string expressions to a floating-point number. This is used here for parameter passing, or in place of defined functions, (which the TS2068

has, FN, but the ZX81 does not). For example, the subroutine MXINV inverts the matrix named in the string X$. If X$="A(I,J)", then MXINV will invert the matrix A(,).

The monitor program provides single keyword entry. I have added a full-size keyboard to my ZX80, and have found that, with practice, it is possible to touch-type these keywords in, saving a lot of time in program entry. Two more very attractive features of Sinclair BASIC are its syntax-checking and the ability to CONTINUE a program even after editing program lines.

The central programs and subroutines can be easily translated to any other BASIC environment, with minor changes. I have translated the matrix-inversion subroutine MXINV to a Commodore VIC20 with no difficulty. However, some things to watch out for when translating: On the ZX81 and TS2068, DIMensioning an array sets all values to 0; VAL on other machines tend to accept only string expressions involving one floating point number only.

Beginning programmers might find the chapter on plotting the most interesting one to start out with. Follow the instructions in the demonstrations, and then experiment with your own variations on the theme. Move on to the last chapter and try out some multiple-precision additions, subtractions, and multiplications. (Try your hand at writing a routine for multiple-precision division).

High-school/college students might use the book as a workbook: to get a better picture of what he is trying to learn in Algebra, Vector-analysis, and Calculus. Use INTEG.2 to numerically integrate those impossible homework problems, (only as a check, of course.)

The person who really needs to find roots to equations, find best-fit curves, or solve differential-equations may find the programs in chapters 6 (Roots and Regression), and 7 (Differential-equations) adequate, or he/she may want to translate them to higher-powered machines.

But to all users of this book: Have a good time experimenting with the programs and subroutines, for it was in that spirit that this book was written. I would like to hear from anyone who has improvements/additions/suggestions/additions/translations/(and anything else.)

USING THE TAPE

The tape supplied with this book holds two copies of the file named "DEMOS", which contains all of the demonstration programs and central subroutines and programs for the TS2068. There is enough space left on the tape to record your own programs.

To use the tape, type

LOAD "DEMOS" or LOAD ""

and start your recorder. The program takes about 2½ minutes to load. Once the program has loaded, it will start automatically. A Title page is first displayed for a few seconds, followed by a menu of all of the demonstrations, (answer "y" to the "scroll?" prompt to get the entire menu listing). Enter your choice by typing the number of the demonstration you want to run. Find the demonstration in the book, and proceed as shown in the "Run program" step.

USING THE BOOK

Each chapter in the book is presented in this format:

1. Introduction
2. Central subroutine(s) or program(s):
   a) Listing(s)
   b) "Spec-sheet(s)"
3. Demonstrations

The "spec-sheet" lists the properties of a particular subroutine or program. It has this format:

1. Short description of what the subroutine or program does.
2. A flow-chart of subroutine or program use:
   a) For a program:
      RUN:
      {program name}: List of prompts, and variables changed.
      STOP: Expected results.
   b) For a subroutine:
      CALL: List of variables which must be initialized or dimensioned before entering subroutine.
      {subroutine name}: List of prompts, and variables changed.
      RETURN: Expected results.
3. Description of algorithm or line-by-line description of program or subroutine.

The demonstrations illustrate the properties of the central subroutines and programs. To use a demonstration, follow the step-by-step instructions to enter the main program, central subroutines, and run the program. The step for running the program has two columns. The idea is to enter the USER ENTERS item followed by an {enter}, and wait for the COMPUTER RESPONDS item. Use the "discussion" and "suggestions" steps for your own programming experiments.

Chapter 1 VECTOR OPERATIONS

Just as some of us do and some of us don't have a direction in life, vectors do and scalars don't have an associated direction.

For the time being, let's assume we are talking about 2-dimensional vectors. Any general (2-d) vector may be described by two "coordinates" x_1 and x_2, the x- and y-components of the vector with respect to some reference system, as shown in figure 1-1. In figure 1-1, the vector is "associated" with the point (a,b). The vector might represent the instantaneous velocity (speed and direction) of a neutron, at a particular time when the neutron is passing through the point (a,b).

The direction of $\vec{X}$ in the figure seems to be about NE, (if N is top). $\vec{X}$ may also be described by two other numbers: its length and its angle with respect to the +X-axis, where

$$\text{LENGTH}(\vec{X}) = \sqrt{x_1^2+x_2^2}$$

$$\text{ANGLE}(\vec{X}) = \text{ATN}(x_2/x_1)$$

Moving up to higher dimensions, it is clear that a set of three coordinates, x_1,x_2,x_3, describes a certain vector in three dimensions. Its length and angles with respect to a given coordinate system may also be defined similarly to the 2-d case. But there are sets of 4,5,...(even ∞) numbers which may also be regarded as vectors in dimensions which can only be imagined.

This chapter presents two subroutines which manipulate the coordinates of vectors. As in the case of the 2-dimensional vector, lengths and directions are determined by use of the subroutines.

The subroutine VINAx finds the minimum and the maximum of a vector's coordinates. This is used in the next chapter by scaling subroutines, which are, in turn, used by plotting subroutines.

The subroutine VDOTP can be used to perform various summation operations. The dot-product between two vectors is one such operation.

Four demonstration problems are given here to demonstrate the vector operations subroutines. In larger programs, it may be advantageous to consolidate the subroutines into the main program. For example, the polynomial regression subroutine, REGRP, found in chapter 6, could have used many calls to the vector operations subroutines; these have been consolidated into one main summation loop.

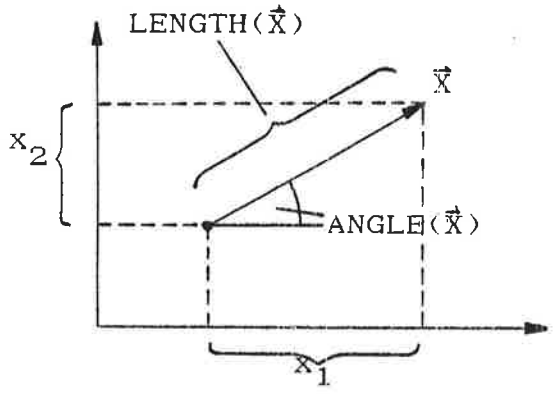


figure 1-1 Vector

```
1000 REM VMINAX (VECTOR MIN,MAX)
1002 LET V=1E37
1004 LET U=-V
1006 FOR I=1 TO N
1008 LET M=VAL X$
1010 IF M<V THEN LET V=M
1012 IF M>U THEN LET U=M
1014 NEXT I
1016 RETURN
1100 REM VDOTP (VECTOR DOT-PRODU
CT)
1102 LET V=0
1104 FOR I=1 TO N
1106 LET V=V+VAL X$
1108 NEXT I
1110 RETURN
```

Figure 1-2 Vector Operations Subroutines

VINAX

This subroutine returns the minimum and maximum element of a vector named by X$.

CALL: X$ = name of vector, indexed by I
N = Dimension of named vector

VINAX: Changes I M U V

RETURN: U = Maximum element of named vector
V = Minimum element of named vector

Example call:

```
LET VINAX=1000
LET N=10
DIM Y(N)
  .
  .
  .
LET X$="Y(I)"
GOSUB VINAX
```

VDOTP

This subroutine returns the dot-product of two named vectors. It may also be used to compute other types of summations, such as the sum of the elements of a vector.

CALL: X$ = string naming summation operation with vectors indexed by I
N = Dimension of named vectors

VDOTP: Changes I V

RETURN: V = Dot-product (or summation)

Example call:

```
LET VDOTP=1100
LET N=10
DIM X(N)
DIM Y(N)
  .
  .
  .
LET X$="X(I)*Y(I)"
GOSUB VDOTP
```

Demonstration 1.1 VECTOR NORMS

Step 1 (PROBLEM): Print the values of these vector norms of a vector $\bar{X}=(X(1),\ldots,X(N))$:

Euclidean norm: $||\bar{X}||_e = \sqrt{\sum_{I=1}^{N} X(I)^2}$

Maximum norm: $||\bar{X}||_m = \max_I(|X(I)|)$

Summation norm: $||\bar{X}||_s = \sum_{I=1}^{N} |X(I)|$

Standard deviation: $\sigma = \sqrt{\frac{1}{N}\sum_{I=1}^{N} (X(I)-M)^2}$; where mean: $M = 1/N \sum_{I=1}^{N} X(I)$

for the vector X = (-1,2,5,3,-1)

Step 2: Enter main program:

```
100 REM DEM01.1
101 LET VINAX=1000
102 LET VDOTP=1100
103 LET N=5
104 DIM X(N)
105 LET X(1)=-1
106 LET X(2)=2
107 LET X(3)=5
108 LET X(4)=3
109 LET X(5)=-1
110 LET X$="X(I)*X(I)"
111 GO SUB VDOTP
112 PRINT "EUCLIDEAN NORM = ";S
QR V
113 LET X$="ABS X(I)"
114 GO SUB VINAX
115 PRINT "MAXIMUM NORM = ";U
116 GO SUB VDOTP
117 PRINT "SUM NORM = ";V
118 LET X$="X(I)"
119 GO SUB VDOTP
120 LET M=V/N
121 LET X$="(X(I)-M)*(X(I)-M)"
122 GO SUB VDOTP
123 PRINT "MEAN = ";M
124 PRINT "STANDARD DEVIATION =
";SQR (V/N)
125 STOP
```

Step 3: Enter subroutines: VINAX VDOTP

Step 4: Run program:

USER ENTERS:	COMPUTER RESPONDS:
RUN	(results printed)

Step 5: Discussion and suggestions:

The norm of a vector can be thought of as its "length", or at least a characteristic scalar value associated with the vector, by which it may be compared with other vectors.

Any norm must obey several criteria:
($||\overline{X}||$ is the norm of the vector $\overline{X}$)

1. $||\overline{X}|| >= 0$; $||\overline{X}|| = 0$ if and only if $\overline{X} = \overline{0}$
2. $||\overline{X}+\overline{Y}|| <= ||\overline{X}|| + ||\overline{Y}||$
3. $||a\overline{X}|| = |a|*||\overline{X}||$

The four norms in this demonstration each assigned different numbers to the vector $\overline{X}=(-1,2,5,3,-2)$, but each still obey the above criteria.

Be sure to try some other vectors besides the particular one given in the demonstration. Try vectors of different dimension also, by changing lines 103-104 to the appropriate dimension). In particular, try the zero vector and thereby verify one of the criteria for vector norms. In fact, try to verify all of the other criteria, by examples.

One problem with this demonstration is that it is not general: the vector dimension and vector coordinates are not changeable without modifying program lines. In the next chapter, a subroutine is presented for entering data points. Using this idea, write a more general demonstration program.

Find some other vector norms, or invent some, and verify the criteria for vector norms by examples.

Demonstration 1.2 SCALAR OPERATIONS BETWEEN TWO VECTORS

Step 1 (PROBLEM): Print the values of these operations between vectors $\vec{X}=(X(1),\ldots,X(N))$ and $\vec{Y}=(Y(1),\ldots,Y(N))$:

Euclidean dot-product: $$(\vec{X},\vec{Y}) = \sum_{I=1}^{N}(X(I)*Y(I))$$

Covariance: $$\mathrm{cov}(\vec{X},\vec{Y}) = \frac{1}{N}\sum_{I=1}^{N}((X(I)-M_X)*(Y(I)-M_y))$$

where: $$M_X = 1/N\sum_{I=1}^{N}X(I)$$

$$M_y = 1/N\sum_{I=1}^{N}Y(I)$$

Distance between $\vec{X}$ and $\vec{Y}$: $$\mathrm{dist}(\vec{X},\vec{Y}) = \sqrt{\sum_{I=1}^{N}(X(I)-Y(I))^2}$$

Angle between $\vec{X}$ and $\vec{Y}$ (in degrees): $$\mathrm{angle}(\vec{X},\vec{Y}) = \mathrm{ACS}\left[\frac{(\vec{X},\vec{Y})}{\sqrt{(\vec{X},\vec{X})*(\vec{Y},\vec{Y})}}\right]*180/\P$$

for the vectors $\vec{X} = (-1,2,5,3,-1)$ and $\vec{Y} = (2,5,7,-1,40)$

Step 2: Enter main program:

```
130 REM DEMO1.2
131 LET VDOTP=1100
132 LET N=5
133 DIM X(5)
134 DIM Y(5)
135 LET X(1)=-1
136 LET X(2)=2
137 LET X(3)=5
138 LET X(4)=3
139 LET X(5)=-1
140 LET Y(1)=2
141 LET Y(2)=5
142 LET Y(3)=7
143 LET Y(4)=-1
144 LET Y(5)=40
145 LET X$="X(I)*Y(I)"
146 GO SUB VDOTP
147 LET XY=V
148 PRINT "(X,Y) = ";XY
149 LET X$="X(I)"
150 GO SUB VDOTP
151 LET MX=V/N
152 LET X$="Y(I)"
153 GO SUB VDOTP
154 LET MY=V/N
155 LET X$="(X(I)-MX)*(Y(I)-MY)
"
156 GO SUB VDOTP
157 PRINT "COV(X,Y) = ";V/N
158 LET X$="(X(I)-Y(I))*(X(I)-Y
(I))"
159 GO SUB VDOTP
160 PRINT "DIST(X,Y) = ";SQR V
161 LET X$="X(I)*X(I)"
162 GO SUB VDOTP
163 LET XX=V
164 LET X$="Y(I)*Y(I)"
165 GO SUB VDOTP
166 LET YY=V
167 PRINT "ANGLE(X,Y) = ";ACS (
XY/SQR (XX*YY))*180/PI;" DEGREES
"
168 STOP
```

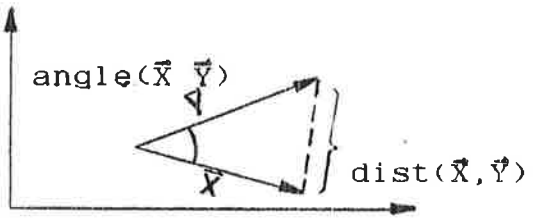
Step 3: Enter subroutines: VDOTP

Step 4: Run program:

USER ENTERS:	COMPUTER RESPONDS:
RUN	(results printed)

Step 5: Discussion and suggestions:

The four operations in this demonstration each assign a different number to the two vectors $\vec{X}$ and $\vec{Y}$. There are two "geometrical" operations: dist($\vec{X}$,$\vec{Y}$), and angle($\vec{X}$,$\vec{Y}$). For two-dimensional vectors, the geometrical picture is this:



Try different vectors and dimensions. Be sure to try some 2- and 3-dimensional vectors to look at the "geometrical" operations for "real-world" vectors.

What do the operations become for 1-dimensional vectors? For example, the Euclidean dot-product degenerates to multiplication for vectors with only one coordinate.

Find some other operations between vectors, or invent some: there are an infinite number of possibilities). All of these operations give a scalar result (a real number, as opposed to an ordered set of real numbers, that is, a vector). There are also operations, such as vector addition, which give a vector result. Write your own subroutines to take care of any operations between two vectors which aren't covered by VINAX or VDOTP.

Finally, the demonstration's main program might be made more general by the addition of a portion for entering dimensions and coordinates of vectors. A routine similar to the subroutine POINT.1, presented in chapter 2, could be added in the place of lines 132 to 144 of the main program.

Demonstration 1.3 DISCRETE CONVOLUTION

Step 1 (PROBLEM): Plot the discrete convolution of $\vec{X}$ and $\vec{Y}$:

$$Y(I) = \begin{cases} 0 \text{ for } I<=0 \\ 1 \text{ for } 0<I<=60 \end{cases}$$

($\vec{Y}$ is a unit step function.)

$$X(I) = \begin{cases} 0 \text{ for } I<=0 \\ 20/I \text{ for } 0<I<=60 \end{cases}$$

Step 2: Enter main program:

```
170 REM DEMO1.3
171 LET VDOTP=1100
172 LET N=60
173 DIM X(N)
174 DIM Y(N)
175 FOR I=1 TO N
176 LET X(I)=20/I
177 LET Y(I)=1
178 NEXT I
179 FOR J=1 TO N
180 LET N=J
181 LET X$="X(I)*Y(J-I+1)"
182 GO SUB VDOTP
183 PLOT J*4,V/3*4: REM *** FOR
ZX81, USE *** PLOT J,V/3
184 NEXT J
185 STOP
```

Step 3: Enter subroutines: VDOTP

Step 4: Run program:

USER ENTERS:	COMPUTER RESPONDS:
RUN (wait about 55s)	(results plotted)

Step 5: Discussion:

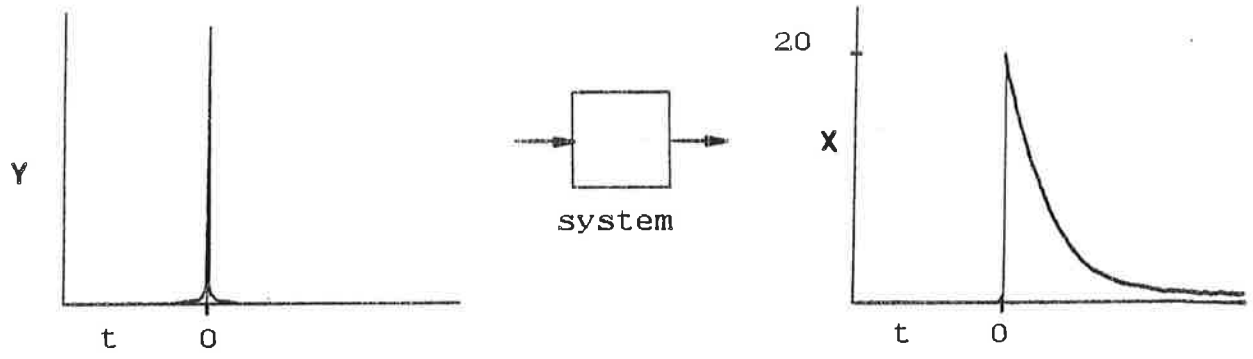
The discrete convolution given in this demonstration is the discrete version of analog convolution. Analog convolution can be described as follows, (the example corresponding to the demonstration).

Suppose some system gives an output signal when it is presented with an input signal. The "system" might be an electronic circuit and the "signals" voltage waveforms. If the system is presented with a unit impulse (a voltage spike in time, with

$$\int_{-\infty}^{\infty} dt\, Y(t) = 1$$

that is, unit area), it responds with the output function

$$X(t) = 20/(t+1)$$



The function X(t) is called the system's impulse response. If the system is presented with another sort of input:

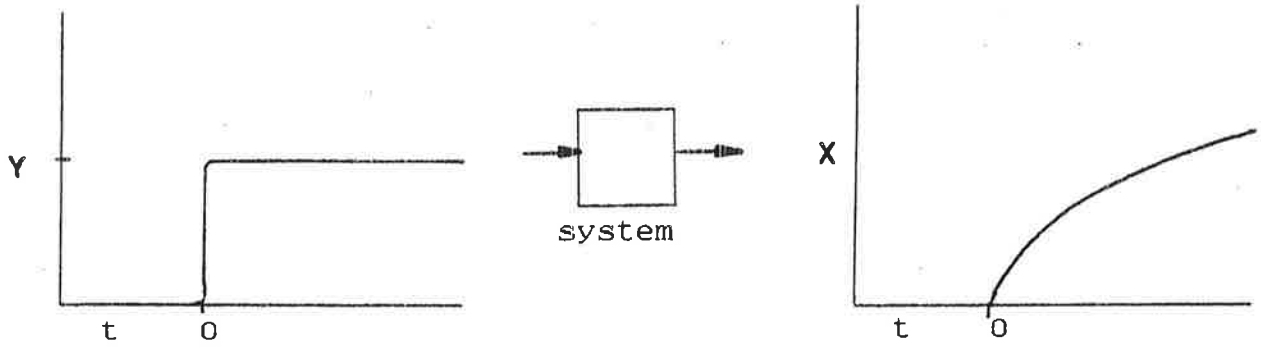
Y(t) = unit step function

the system's response will be the step-response of the system. In general, the output of the system will be the result of convolving the input signal with the system's impulse response:

$$\text{Convolution}(X,Y) = \int_{-\infty}^{\infty} X(s)Y(t-s)ds$$

So the step-response is given by:

$$\int_{0}^{t} 20ds/(s+1) = 20*LN(t+1)$$



Step 6: Suggestions

It may be possible to generalize this program to accept functions in the form of strings. See chapter 6 for some hints.

The function X(I) may be varied by changing the constant 20 in line 176 of the main program. See what effect this has on the results. Try other impulse response functions, X(I), and system input functions, Y(I).

Demonstration 1.4: GRAM-SCHMIDT ORTHONORMALIZATION

Step 1 (PROBLEM): Find an orthonormal basis for the vectors $\vec{X}_1$, $\vec{X}_2$, and $\vec{X}_3 \in \mathbb{R}^5$:

$\vec{X}_1 = (1,1,1,1,1)$

$\vec{X}_2 = (0,1,1,1,1)$

$\vec{X}_3 = (0,0,1,1,1)$

Step 2: Enter main program:

```
190 REM DEMO1.4
191 LET VDOTP=1100
192 LET N=5
193 LET P=3
194 DIM X(N,P)
195 FOR J=1 TO P
196 FOR I=J TO N
197 LET X(I,J)=1
198 NEXT I
199 NEXT J
200 FOR J=1 TO P
201 FOR K=1 TO J-1
202 LET X$="X(I,J)*X(I,K)"
203 GO SUB VDOTP
204 FOR I=1 TO N
205 LET X(I,J)=X(I,J)-V*X(I,K)
206 NEXT I
207 NEXT K
208 LET X$="X(I,J)*X(I,J)"
209 GO SUB VDOTP
210 LET M=SQR V
211 FOR I=1 TO N
212 LET X(I,J)=X(I,J)/M
213 NEXT I
214 NEXT J
215 FOR J=1 TO P
216 FOR I=1 TO N
217 PRINT "E";J;"(";I;")= ";
218 PRINT SGN X(I,J)*INT (ABS X
(I,J)*1E6)*1E-6
219 NEXT I
220 PRINT "**********"
221 NEXT J
222 STOP
```

Step 3: Enter subroutines: VDOTP

Step 4: Run program:

USER ENTER:	COMPUTER RESPONDS:
RUN (wait about 10s)	(results printed)

Step 5: Discussion:

This demonstration finds a set of basis vectors $\{\vec{E}_I\}$, 5-dimensional vectors, which have the following properties:

1. Each $\vec{E}_I$ has a Euclidean norm of 1 (see demonstration 1.1).
2. The Euclidean dot-product $(\vec{E}_I,\vec{E}_J) = 1$ if I=J or 0 if I<>J. This means that the basis vectors are orthogonal to one another, the angle between any two is 90 degrees (see demonstration 1.2).
3. Any vector in the vector-space $\mathbb{R}^5$, the collection of all 5-dimensional vectors, which can be written as a linear sum of $\vec{X}_1$, $\vec{X}_2$, and $\vec{X}_3$ (given in the demonstration), may also be written as a linear sum of the basis vectors $\{\vec{E}_I\}$. That is, if $\vec{Y}$ can be written as the sum:

$$\vec{Y} = a_1\vec{X}_1 + a_2\vec{X}_2 + a_3\vec{X}_3$$

Then there also exist b's such that $\vec{Y}$ can also be written:

$$\vec{Y} = b_1\vec{E}_1 + b_2\vec{E}_2 + b_3\vec{E}_3$$

Note that, in the print-out section of the main program, lines 215 to 221, the results are truncated (line 218). There still may be trouble for some other vectors if the truncation error is less than the actual coordinate value.

In addition, the Gram-Schmidt method requires that the vectors $\{\vec{X}_I\}$ be linearly-independent. That is, no one of the $\vec{X}$'s may be written as the linear sum of the rest of the $\vec{X}$'s.

Step 6: Suggestions:

Lines 192 to 199 might be replaced with a routine for entering any set of vectors from the keyboard.

Analyze the workings of the program with a flowchart, and note any difficulties which might occur with the algorithm for certain cases. For example, what if the number of vectors is greater than the dimension of the vector-space? This set, of course, would be a linearly-dependent one. If possible, correct some of the deficiencies in the program.

as always, try other vectors than those given in the demonstration. In particular, let one of the $\vec{X}$'s be the zero vector. (The zero vector can always be written as the linear sum of any other set of vectors).

Chapter 2 PLOTTING

There comes a time in everyone's life when he or she must deal with data. The person who wishes to have a picture of what his data is doing plots it. It is to this person that this chapter is addressed. In addition, any person who finds watching patterns a worthwhile activity should listen in.

The subroutine PLOTD plots data points, which are input via the subroutines POINT.1 or POINT.2. PLOTD also makes use of SCALE.1 or SCALE.2 to scale the data vectors so that all of the data points fit within the plot. PLOTD assumes that the data it is given is cartesian in nature, (x- and y-coordinate). Once it is called, it finds an appropriate plotting window via SCALE.1, or SCALE.2, and then plots the data found in vectors X() and Y() in the form (X(I),Y(I)). The difference between SCALE.1 and SCALE.2 is that SCALE.1 finds endpoints which may be expressible as an integer times a power of ten, where SCALE.2 finds the minimum and maximum of the data vector, (X(), or Y()).

Data is supplied to PLOTD via one of two subroutines, POINT.1 and POINT.2. POINT.1 prompts the user (you) for the number of points and each data point, (in which case you may either respond or go back to sleep.) POINT.2 prompts the user for an expression of X for sampling at equal intervals. POINT.2 also asks for how many points to be gathered, and over what range of X values to sample.

Demonstrations 2.1, 2.2, and 2.3 illustrate the uses of these subroutines. In addition, demonstrations in other chapters make use of some of these subroutines for their particular purposes. The demonstrations in this chapter plot cartesian data, polar data and a sampled function.

The program PLOT2 is a program that plots plane-curves which are described by any of 4 different types of curve descriptions: cartesian, cartesian parametric, polar, and polar parametric. The user supplies PLOT2 with an expression or expressions to be used to generate a plot, the desired plotting window, and for parametric and polar plots, a range for the parameter to be swept over.

The program PLOT3 generates a picture of the contours of a three-dimensional surface. The user supplies an expression in X and Y, a plotting window, and a parameter called the contour-spacing-parameter, (C-S-P). The C-S-P determines how far apart to space the contours.

Example curves and contours are given, for comparison with results from the demonstrations. There is a lot of room for experimentation in this chapter. Changing the value of a constant or two in the expressions of the demonstrations can give very different results.

Plotting usually takes considerable amounts of time, and you may want to do it in conjunction with another activity, such as writing a letter to your congressman. For example, it took about

13 minutes to plot the contours of the magnitude of the complex sine function, (on the ZX81). Not all of the plots in the demonstrations in this chapter take quite that long. However, be patient! Using a computer to do your plotting is still much faster and easier than doing it by hand.

```
2000 REM POINT.1 (DATA INPUT)
2002 PRINT "INPUT NO. PTS."
2004 INPUT N
2006 DIM X(N)
2008 DIM Y(N)
2010 FOR I=1 TO N
2012 IF INT (I/15)*15=I THEN CLS
2014 PRINT "X(";I;"),Y(";I;") = ?
 ";
2016 INPUT X(I)
2018 INPUT Y(I)
2020 PRINT "(";X(I);",";Y(I);")"
2022 NEXT I
2024 RETURN
2100 REM POINT.2 (SAMPLE FUNCTIO
N)
2102 PRINT "INPUT NO. PTS."
2104 INPUT N
2106 DIM X(N)
2108 DIM Y(N)
2110 PRINT "INPUT Y EXPRESSION=Y
(X)"
2112 INPUT Y$
2114 PRINT "INPUT XMIN,XMAX"
2116 INPUT A
2118 INPUT B
2120 FOR I=1 TO N
2122 LET X=(B-A)*(I-1)/(N-1)+A
2124 LET X(I)=X
2126 LET Y(I)=VAL Y$
2128 NEXT I
2130 RETURN
2200 REM SCALE.1 (PLOTTING WINDO
W 1)
2202 GO SUB VINAX
2204 LET M=U-V+(V=U)
2206 LET P=M
2208 IF ABS V>P THEN LET P=ABS V
2210 IF ABS U>P THEN LET P=ABS U
2212 LET P=10↑INT (LN P/LN 10)
2214 LET V=INT (V/P)*P
2216 LET M=INT (M/P)*P+V
2218 IF M>=U THEN GO TO 2224
2220 LET M=M+P
2222 GO TO 2218
2224 LET U=M*(ABS M>1E-6)
2226 LET V=V*(ABS V>1E-6)
2228 RETURN
2300 REM SCALE.2 (PLOTTING WINDO
W 2)
2302 GO SUB VINAX
2304 IF U<>V THEN RETURN
2306 LET V=INT V
2308 LET U=V+1
2310 RETURN
2400 REM PLOTD (PLOT DATA PTS.)
2402 LET X$="X(I)"
2404 GO SUB SCALE
2406 LET A=V
2408 LET B=U
2410 LET X$="Y(I)"
2412 GO SUB SCALE
2414 LET C=V
2416 LET D=U
2418 CLS
2420 FOR I=1 TO N
2422 PLOT 240*(X(I)-A)/(B-A),160
*(Y(I)-C)/(D-C): REM *** FOR ZX8
1, USE *** PLOT 80*(X(I)-A)/(B-A
),40*(Y(U)-C)/(D-C)
2424 NEXT I
2426 PRINT AT 0,0;"FROM (";A;","
;C;") TO (";B;",";D;")"
2428 RETURN
```


Figure 2-1 Data point input, bounding, and plotting subroutines

POINT.1

This subroutine prompts the user for data points. The number of data points is first entered, then each point is entered into vectors X() and Y().

CALL:

POINT.1 Changes I N X() Y()
Prompts, in order of appearance:
1. "INPUT NO. PTS."
2. "X(I),Y(I)= ? "

RETURN: X() = x-coordinates of each data point
Y() = y-coordinates of each data point
N = dimension of X() and Y() = number of data points

Example call:

```
LET POINT=2000
GOSUB POINT
```

Summary of program operation:

Lines 2002 to 2008: prompt user for number of points, and dimension arrays X(N) and Y(N).
Lines 2010 to 2022: prompt user for each point and enter values into X(I) and Y(I).

POINT.2

This subroutine prompts the user for an expression to be sampled at equally-spaced intervals. The expression is entered in the form of a string variable, as a function of X; for example Y(X)="X*2+3/X". The number of data points is first entered, then the expression is entered, and then the range of X-values for sampling is entered. The subroutine then samples the function and enters the x- and y-coordinates into the vectors X() and Y().

CALL:

```
POINT.2    Changes A B N X() Y() Y$
           Prompts, in order of appearance:
           1. "INPUT NO. PTS."
           2. "INPUT Y EXPRESSION=Y(X)"
           3. "INPUT XMIN,XMAX"

RETURN:    X() = x-coordinates of each data point
           Y() = y-coordinates of each data point
           N = dimension of X() and Y() = number of data points
           A = Xmin
           B = Xmax
           Y$ = Y expression, Y(X)
```

Example call:

```
LET POINT=2100
GOSUB POINT
```

Summary of operation:

Lines 2102 to 2108: prompt user for number of data points, and dimension X(N) and Y(N).
Lines 2110 to 2118: prompt user for a function of X, and Xmin,Xmax.
Lines 2120 to 2128: sample the function and enter the samples into X(I) and Y(I).

SCALE.1

This subroutine finds lower and upper bounds for a vector named in X$. The lower and upper bounds are each an integer times a power of ten.

CALL: VINAX = 1000
X$ = Name of vector to be bounded, indexed by I
N = Dimension of vector

SCALE.1: Changes I M P U V
Calls VINAX

RETURN: V = Lower bound
U = Upper bound

Example call:

```
LET VINAX=1000
LET SCALE=2200
LET N=10
DIM X(N)
  .
  .
  .
LET X$="X(I)"
GOSUB SCALE
```

Summary of operation:

Line 2202: determine minimum and maximum values of data vector named by string X$. For example, if X$="X(I)" then the minimum and maximum of X() are found.
Lines 2204 to 2212 provide for the case that the minimum and maximum of the data vector are the same. In this case, the returned lower and upper bounds are separated by 1.
Lines 2214 to 2228 determine the lower endpoint.
Lines 2230 to 2238 determine the upper endpoint.

SCALE.2

This subroutine finds lower and upper bounds for a vector named in X$. The lower and upper bounds are the minimum and maximum values of the data vector.

CALL: VINAX = 1000
X$ = Name of vector to be bounded, indexed by I.
N = Dimension of vector

SCALE.2: Changes I M U V
Calls VINAX

RETURN: V = Lower bound
U = Upper bound

Example call:

```
LET VINAX=1000
LET SCALE=2300
LET N=10
DIM X(N)
  :
  :
LET X$="X(I)"
GOSUB SCALE
```

Summary of operation:

Line 2302: Determine minimum and maximum values of the data vector named by string X$.
Lines 2304 to 2312 provide for the case that all values of the data vector are the same. The returned lower and upper bounds, in this case, are separated by 1.

PLOTD

This subroutine plots data in the vectors X() and Y(). X() contains the x-coordinates and Y() contains the y-coordinates of each data point. For example, data point number 5 is :

```
x = X(5)  ;  y = Y(5)
```

CALL: SCALE = 2200 for SCALE.1 or 2300 for SCALE.2
VINAX = 1000
X() = x-coordinates of each data point
Y() = y-coordinates of each data point
N = Number of data points

PLOTD: Changes A B C D G I M N P U V X() X$ Y()
Calls SCALE.1 or SCALE.2 (which call VINAX)

RETURN: Data plotted
Lower left and upper right endpoints of plotting-window printed at top of screen

Example call:

```
LET PLOTD=2400
LET VINAX=1000
LET POINT=2000
LET SCALE=2200
GOSUB POINT
GOSUB PLOTD
```

Summary of operation:

Lines 2402 to 2416: Determine plotting window.
Lines 2418 to 2424: Plot each data point.
Line 2426: print lower left and upper right end-points of the plotting window.

Demonstration 2.1 PLOT CARTESIAN DATA

Step 1: Enter main program:

```
230 REM DEMO2.1
231 LET POINT=2000
232 LET VINAX=1000
233 LET SCALE=2200
234 LET PLOTD=2400
235 GO SUB POINT
236 GO SUB PLOTD
237 STOP
```

Step 2: Enter subroutines: VINAX POINT.1 SCALE.1 PLOTD

Step 3 (PROBLEM): Plot the cartesian data in the table below:

I:	X(I):	Y(I):
1	0	0
2	-1	-1.6
3	-.36	-.4
4	.2	.14
5	1	.5
6	1.7	.38
7	2.1	.2
8	3.5	0

Step 4: Run program:

USER ENTERS:	COMPUTER RESPONDS:
RUN	INPUT NO. PTS.
8	X(1),Y(1)=?
0	
0	X(2),Y(2)=?

... enter data from table in step 3 ...

.2	X(8),Y(8)=?
3.5	
0	(results plotted)

Step 5: Discussion of results

The data in this example was fabricated. Any resemblance to actual data is purely coincidental.

Line 231 of the main program selected POINT.1 to fill the data vectors X() and Y(), so that you could hand enter the data. SCALE.1 found two endpoints for each data vector so that all of the data would appear on the plot. Since SCALE.1 was used, the two endpoints for each axis should be integers times a power of 10. In this case:

Xmin = -1 ; (= $-1*10^0$)
Xmax = 4
Ymin = -2
Ymax = 1

The lower left and upper right endpoints of the plot are printed above the plot. In this case:

Lower left: (Xmin,Ymin) = (-1,-2)
Upper right: (Xmax,Ymax) = (4,1)

PLOTTING WINDOW:

Step 6: Suggestions

Since you have spent some time entering the subroutines and main program, SAVE them onto a file for future use. You may want to keep the subroutines in memory for the time being and try demonstrations 2.2 and 2.3.

This demonstration can be used to plot any other cartesian data you may have lying around. Hint: look in a magazine for an article on the economy. Chapter 6 also contains some tables of data which are used in the demonstrations there. If you don't have any data on hand, make some up and plot it. Generate some figures in this way.

For a variation on this demonstration, try to write a main program which makes use of PLOTD, or a similar subroutine, and which plots one data point at a time. The plotting window would have to be pre-determined, and SCALE.1 and SCALE.2 subroutines wouldn't be needed. Since PLOTD calls SCALE automatically, you can get around this by writing a subroutine SCALE.3:

```
2700 REM SCALE.3
2702 LET U={pre-determined value}
2704 LET V={pre-determined value}
2706 IF X$="X(I)" THEN RETURN
2708 LET U={pre-determined value}
2710 LET V={pre-determined value}
2712 RETURN
```

Or more simply, edit out some of the lines in PLOTD.

For another variation, try to combine the main program and subroutines all into one big program.

Demonstration 2.2 PLOT POLAR DATA

Step 1: Enter main program:

```
240 REM DEMO2.2
241 LET POINT=2000
242 LET VINAX=1000
243 LET SCALE=2300
244 LET PLOTD=2400
245 GO SUB POINT
246 FOR I=1 TO N
247 LET M=X(I)*PI/180
248 LET X(I)=Y(I)*COS M
249 LET Y(I)=Y(I)*SIN M
250 NEXT I
251 GO SUB PLOTD
252 STOP
```

Step 2: Enter subroutines: VINAX POINT.1 SCALE.2 PLOTD

Step 3 (PROBLEM): Plot the polar data in the table below:

I:	θ(I):	R(I):
1	0°	0
2	15°	.2
3	45°	.6
4	50°	.5
5	70°	.2
6	75°	.1
7	80°	0

Step 4: Run program:

USER ENTERS:	COMPUTER RESPONDS:
RUN	INPUT NO. PTS.
7	X(1),Y(1)=?
0	
0	X(2),Y(2)=?

... enter data from table in step 3 ...

.1	X(7),Y(7)=?
80	
0	(results plotted)

Step 5: Discussion:

You should have obtained a plot similar to figure 2-2, below. This is a plot of the points in the table on polar graph paper.

The demonstration is an example where, after data vectors X() and Y() have been filled via POINT.1, the data is manipulated (lines 246 to 250) before it is submitted to PLOTD for plotting. Thus, in this example, the polar data entered via POINT.1 is converted to cartesian data, which is the appropriate form for PLOTD. Note that in line 247, a degree to radian conversion is made, (see Appendix **A**).

Line 243 selected SCALE.2 for finding appropriate endpoints for the X and Y axes. The endpoints should be the minima and maxima of the data vectors X() and Y().

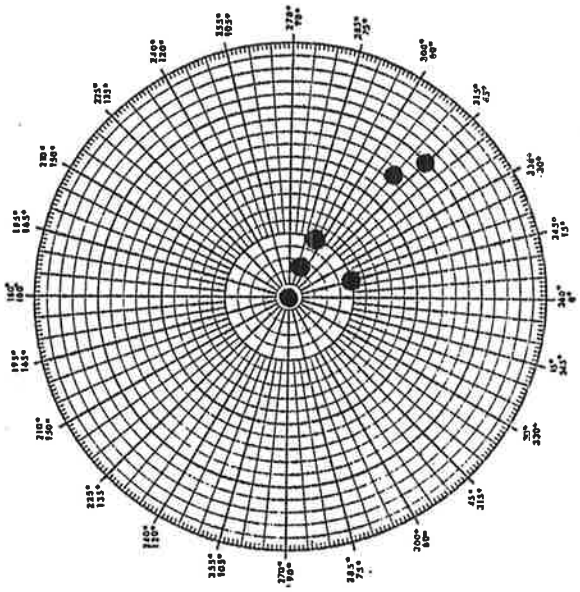


Figure 2-2 Polar plot

Step 6: Suggestions:

If you have need of a polar data plotting program from time to time, SAVE the main program and subroutines in this demonstration. Before you throw away what you have done with a NEW, try plotting some different polar data. Create some data of your own or use a function from table 2-1 to generate a few points. You may want to keep the subroutines in memory for the next demonstration.

For a variation of this example, alter the main program so that it plots polar data in the form of radians and radii.

You can insert another type of data manipulation routine before plotting the results. For example, a derivative-taking subroutine, DERIV.1 is found in chapter 4. The demonstration of DERIV.1, demonstration 4.2 differentiates the data before plotting.

For another variation, combine the program and some or all of the subroutines into one main program.

Demonstration 2.3 PLOT SAMPLED FUNCTION AS POLAR DATA

Step 1: Enter main program:

```
260 REM DEMO2.3
261 LET POINT=2100
262 LET VINAX=1000
263 LET SCALE=2300
264 LET PLOTD=2400
265 GO SUB POINT
266 FOR I=1 TO N
267 LET M=X(I)
268 LET X(I)=Y(I)*COS M
269 LET Y(I)=Y(I)*SIN M
270 NEXT I
271 GO SUB PLOTD
272 STOP
```

Step 2: Enter subroutines: VINAX POINT.2 SCALE.2 PLOTD

Step 3 (PROBLEM): Sample the function below and plot as polar data:

$$R(\theta) = \cos(\theta) \quad ; \quad 0 <= \theta <= \pi$$

Step 4: Run program:

USER ENTERS:	COMPUTER RESPONDS:
RUN	INPUT NO. PTS.
200	INPUT Y EXPRESSION=Y(X)
COS X	INPUT XMIN,XMAX
0	
PI (wait about 1m)	(results plotted)

Step 5: Discussion and suggestions:

You should have obtained a plot of a unit circle with center at (.5,0).

The program PLOT2, listed in figure 2-3, can perform the same function as this demonstration program: plotting a polar function. However, the use of PLOTD does have the advantage in that after a function has been sampled, the data may be manipulated before submitting it to be plotted.

In this example, POINT.2 has been used to sample the function at equal intervals, and SCALE.2 has been used to give the maximum plotting resolution (SCALE.1 would have found a somewhat larger plotting window).

You will find more examples of polar functions to sample in table 2-1. Try plotting one or two of them.

For a variation on the theme of this demonstration, look at demonstration 4.2, where some data has been differentiated before sending it off to be plotted.

```
2500 REM PROGRAM PLOT2 (PLOT CUR
VES)
2502 PRINT "CART,CART.PARA.,POLA
R,OR POLAR PARA.?"
2504 PRINT "INPUT 1,2,3,OR 4"
2506 INPUT Z
2508 PRINT "PLOTTING WINDOW?"
2510 PRINT "INPUT XMIN,XMAX,YMIN
,YMAX"
2512 INPUT A
2514 INPUT B
2516 INPUT C
2518 INPUT D
2520 LET E=A
2522 LET F=B
2524 IF Z=1 THEN GO TO 2542
2526 PRINT "PARAMETER/THETA RANG
E?"
2528 PRINT "INPUT TMIN,TMAX"
2530 INPUT E
2532 INPUT F
2534 IF Z=3 THEN GO TO 2542
2536 PRINT "X/THETA EXPRESSION?"
2538 PRINT "INPUT X(T) OR THETA
(T)"
2540 INPUT X$
2542 PRINT "Y/R EXPRESSION?"
2544 PRINT "INPUT Y(X) OR Y(T) O
R R(T)"
2546 INPUT Y$
2548 CLS
2550 FOR T=E TO F STEP (F-E)/200
2552 LET X=T
2554 IF Z=2 OR Z=4 THEN LET X=VA
L X$
2556 LET Y=VAL Y$
2558 IF Z<3 THEN GO TO 2566
2560 LET M=X
2562 LET X=Y*COS M
2564 LET Y=Y*SIN M
2566 IF X<A THEN LET X=A
2568 IF X>B THEN LET X=B
2570 IF Y<C THEN LET Y=C
2572 IF Y>D THEN LET Y=D
2574 PLOT 255*(X-A)/(B-A),175*(Y
-C)/(D-C): REM *** FOR ZX81, USE
 *** PLOT 63*(X-A)/(B-A),43*(Y-C
)/(D-C)
2576 NEXT T
2578 STOP
```

Figure 2-3 Plane curve plotting program

PLOT2

This program plots plane curves described by one of the following representations:

1. Cartesian expression: $Y = F(X)$
2. Cartesian parametric expressions: $X = F_1(T)$; $Y = F_2(T)$
3. Polar expression: $R = F(T)$
4. Polar parametric expressions: $R = F_1(T)$; $\theta = F_2(T)$

RUN:

PLOT2: Changes A B C D E F M T X X$ Y Y$ Z
Prompts, in order of appearance:

1. "CAR.,CART.PARA.,POLAR,or POLAR PARA.?"
   "INPUT 1,2,3,OR 4"
2. "PLOTTING WINDOW?"
   "INPUT XMIN,XMAX,YMIN,YMAX"
3. "PARAMETER/THETA RANGE?"
   "INPUT TMIN,TMAX"
4. "X/THETA EXPRESSION?"
   "INPUT X(T) OR THETA(T)"
5. "Y/R EXPRESSION?"
   "INPUT Y(X) OR Y(T) OR R(T)"

STOP: Curve plotted

Summary of operation:

Lines 2500 to 2546: Prompt user for type of curve, plotting window, independent variable range, and X/THETA and Y/R expressions.
Line 2550: Compute step-size and limits for sweep of independent variable, (X or T).
Lines 2552 to 2556: Assign values to X and Y for each value of T.
Lines 2558 to 2564: In the case of polar plots, convert from polar coordinates to cartesian coordinates.
Lines 2566 to 2573: Limit coordinates to plotting window.
Line 2574: Convert from cartesian coordinates to pixel coordinates, and PLOT.

Demonstration 2.4 PLOT SEVERAL CURVES FROM TABLE 2-1

Step 1: Enter program: PLOT2

Step 2 (PROBLEM 1): Plot the serpentine with the cartesian form

$$Y = X/[(X/6)^2+1/9]$$

This is shown in figure 2-4 .
Use the plotting window: (Xmin,Ymin) = (-10,-10)
(Xmax,Ymax) = (10,10)

Step 3: Run program:

USER ENTERS:	COMPUTER RESPONDS:
GOTO 2500	CART.,CART.PARA.,POLAR,OR POLAR PARA.?
	INPUT 1,2,3,OR 4
1	PLOTTING WINDOW?
	INPUT XMIN,XMAX,YMIN,YMAX
-10	
10	
-10	
10	Y/R EXPRESSION?
	INPUT Y(X) OR Y(T) OR R(T)
X/(X*X/36+1/9)	
{wait about 20s}	(results plotted)

Step 4 (PROBLEM 2): Plot the Lissajous pattern with the cartesian parametric form

$$X = SIN(T)$$
$$Y = SIN(3*T/4)$$

for $-20<=T<=20$.
This is shown in figure 2-5 .
Use the plotting window: (Xmin,Ymin) = (-1,-1)
(Xmax,Ymax) = (1,1)

Step 5: Run program:

USER ENTERS:	COMPUTER RESPONDS:
GOTO 2500	CART.,CART.PAR.,POLAR,OR POLAR PARA.?
	INPUT 1,2,3,OR 4
2	PLOTTING WINDOW?
	INPUT XMIN,XMAX,YMIN,YMAX
-1	
1	
-1	
1	PARAMETER/THETA RANGE?
	INPUT TMIN,TMAX

-20 20	X/THETA EXPRESSION? INPUT X(T) OR THETA(T)
SIN T	Y/R EXPRESSION? INPUT Y(X) OR Y(T) OR R(T)
SIN (.75*T) (wait about 35s)	(results plotted)

Step 6 (PROBLEM 3): Plot the Limaçon of Pascal with the polar equation

$$R = 2*COS(\theta) + 1$$

for $-\pi <= \theta <= \pi$
Use the plotting window: (Xmin,Ymin) = (-.5,-2)
(Xmax,Ymax) = (3,2)

Step 7: Run program:

USER ENTERS:	COMPUTER RESPONDS:
GOTO 2500	CART.,CART.PARA.,POLAR,OR POLAR PARA.? INPUT 1,2,3,OR 4
3	PLOTTING WINDOW? INPUT XMIN,XMAX,YMIN,YMAX
-.5 3 -2 2	PARAMETER/THETA RANGE? INPUT TMIN,TMAX
-PI PI	Y/R EXPRESSION? INPUT Y(X) OR Y(T) OR R(T)
2*COS T+1 (wait about 45s)	(results plotted)

Step 8 (PROBLEM 4): Plot the Rhodonea with polar parametric equations

$$R = COS(T)$$
$$\theta = T/7$$

for $-\pi <= T <= \pi$
Use the plotting window: (Xmin,Ymin) = (-1,-1)
(Xmax,Ymax) = (1,1)

Step 9: Run program:

USER ENTERS:	COMPUTER RESPONDS:
GOTO 2500	CART.,CART.PARA.,POLAR,OR POLAR PARA.? INPUT 1,2,3,OR 4
4	PLOTTING WINDOW? INPUT XMIN,XMAX,YMIN,YMAX
-1 1 -1 1	PARAMETER/THETA RANGE? INPUT TMIN,TMAX
-4*PI 4*PI	X/THETA EXPRESSION? INPUT X(T) OR THETA(T)
T/7	Y/R EXPRESSION? INPUT Y(X) OR Y(T) OR R(T)
COS T {wait about 30s}	(results plotted)

Step 10: Discussion:

A serpentine, some Lissajous patterns, a Limaçon of Pascal, and two Rhodoneae are plotted in figures 2-**4** and 2-**5** . Compare your results, keeping in mind the limited graphics resolution of the ZX81. Plotting times, which are typically long, have been given in each problem.

If you followed the demonstration carefully, you should have had no trouble entering the plotting window. However, when you begin to experiment beyond the demonstration, remember to enter XMIN,XMAX, YMIN,and YMAX in that order (as prompted). Also, remember that expressions must contain the appropriate independent variable: X in cartesian expressions, T in polar and parametric expressions.

Values of constants used in expressions may be entered into variables used in expressions in immediate mode before executing the program. For example, if you have already assigned values to the variables YINT and SLOPE, then the expression

"Y(X)=SLOPE*X+YINT"

is a valid expression to use. (Remember to GOTO 2500, not RUN, when doing this).

Step 11: Suggestions:

Using PLOT2 should become easier the more you use it. So try some more examples! Use equations and plotting parameters from table 2-1 and figures 2-**4** and 2-**5** . Keep track of your plotting parameters for each plot.

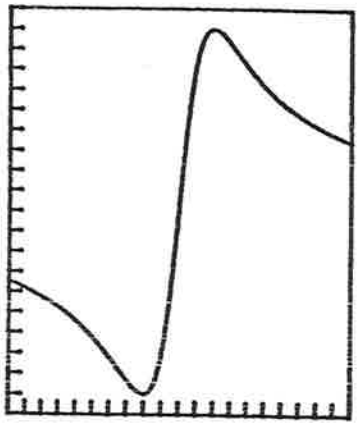
There is much room for experimentation in this demonstration. Vary the constants in the function of each problem for different

results. "Warp" the functions; for example, change a SIN(T) to a COS(T), or even EXP(T).

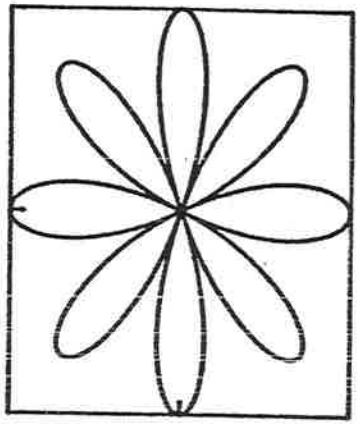
One bothersome thing about PLOT2 is that it takes so darned long to plot a function. Try experimenting to get faster plots. You might try plotting fewer points than 200 (line 2550), or leaving out some of the branching or window clipping (lines 2566-2572), or change line2574 so that the program doesn't have to think so hard for each point.

If you have access to a higher power machine, try to convert this program over just to see how much faster it will run.

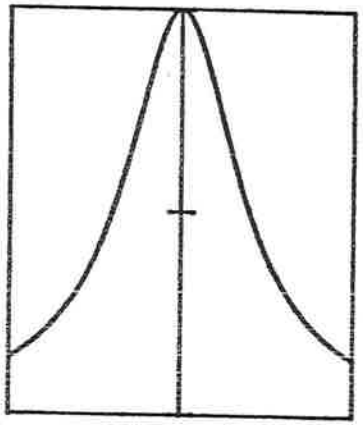
The program may be split up into 4 different programs, each for a different curve representation. Some savings in time may be obtained in this way also.



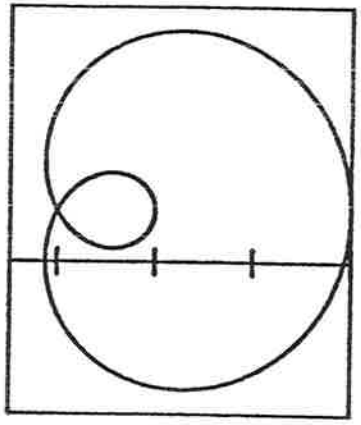
(a)



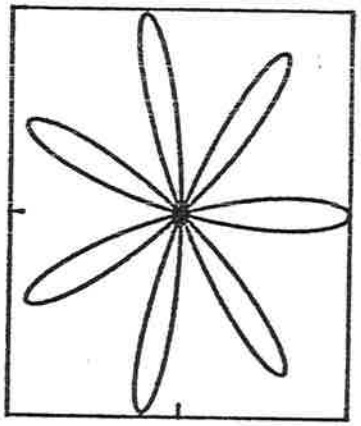
(c)



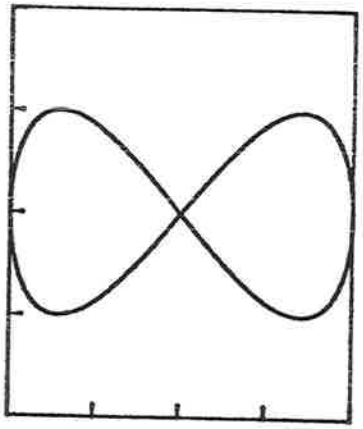
(e)



(b)



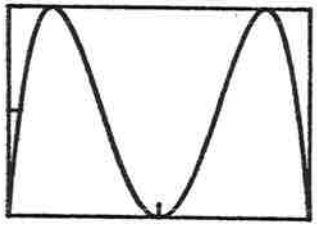
(d)



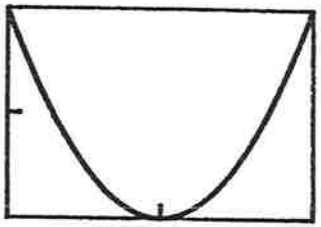
(f)

Figure 2-4 Plane curves

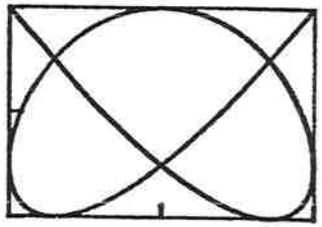
	Name:	Equations of curve:	T-range:	Plotting window:
a	Serpentine	$Y=X/[(X/6)^2+1/9]$		(-10,-10),(10,10)
b	Limaçon of Pascal	$R=2COS(T)+1$, $\theta=T$	[-¶,¶]	(-1.5,-2),(2,2)
c	Rhodonea	$R=COS(4T)$, $\theta=T$	[-¶,¶]	(-1,-1),(1,1)
d	Rhodonea	$R=COS(7T)$, $\theta=T$	[-¶,¶]	(-1,-1),(1,1)
e	Witch of Agnesi	$Y=2/[(X/2)^2+1]$		(-1,0),(1,2)
f	Eight curve	$R=2COS(T)*\ 1+SIN^2(T)$ $\theta=ATN(SIN(T))$	[-¶,¶]	(-2,-2),(2,2)



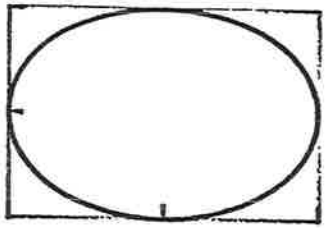
(d)



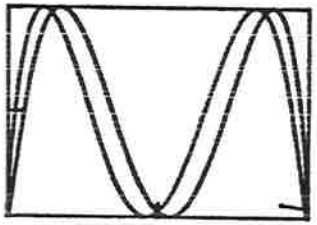
(h)



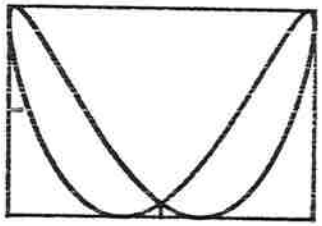
(l)



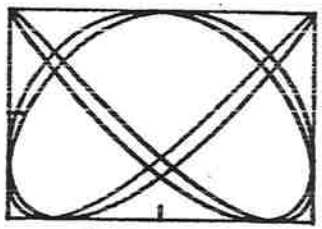
(p)



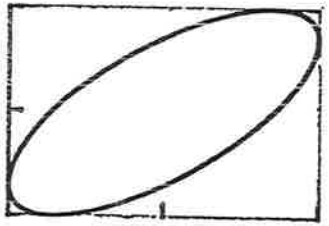
(c)



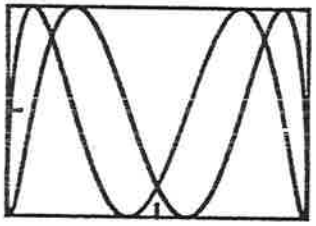
(g)



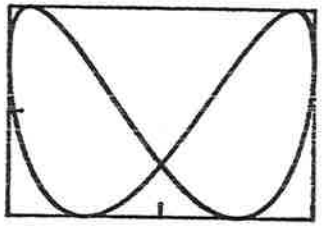
(k)



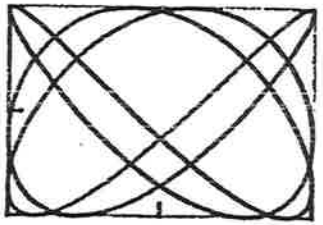
(o)



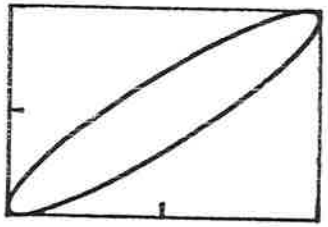
(b)



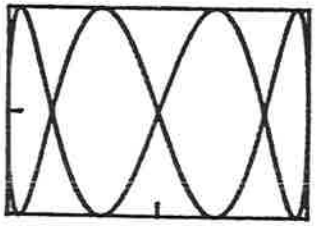
(f)



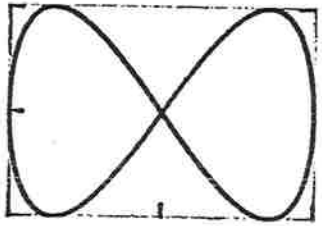
(j)



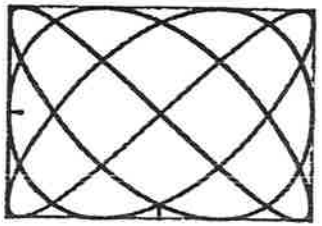
(n)



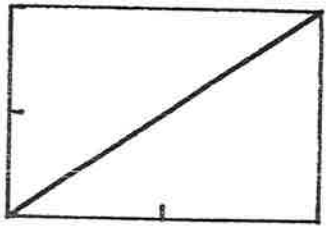
(a)



(e)



(i)



(m)

Figure 2-5 Lissajous patterns:

The equations used for these plots all have the form:

```
X(T)=SIN(aT+b)
Y(T)=SIN(T)
```

where a and b are the constants given in the table below.
The independent variable, T, was varied over the T-range given in the table, and the plotting window in each case was

```
(Xmin,Ymin) = (-1,-1)
(Xmax,Ymax) = (1,1)
```

Section:	a:	b:	T-range:
a	.25	0	[-20,20]
b	.25	¶/16	[-20,20]
c	.25	¶/10	[-20,20]
d	.25	¶/8	[-20,20]
e	.5	0	[-10,10]
f	.5	¶/12	[-10,10]
g	.5	¶/6	[-10,10]
h	.5	¶/4	[-10,10]
i	.75	0	[-20,20]
j	.75	¶/16	[-20,20]
k	.75	¶/10	[-20,20]
l	.75	¶/8	[-20,20]
m	1	0	[-4,4]
n	1	¶/8	[-4,4]
o	1	¶/4	[-4,4]
p	1	¶/2	[-4,4]

Table 2-1.a Cartesian forms. a,b,c,d = arbitrary constants.

Name:	Y(X):	X-range:
Hyperbola	a/X	(-∞,∞) ≠0
Witch of Agnesi	a/((X/a)**2+1)	(-∞,∞)
Serpentine	X/((X/a)**2+b**2)	(-∞,∞)
Quatrix of Hippias	X/TAN(a*X)	(0,¶/a)
Trident of Newton	a/X+b+c*X+d*X**2	(-∞,∞) ≠0
Polynomial	$\sum_{n=0}^{N} a_n$*X**n	(-∞,∞)
Normal Error Curve	a/EXP(¶*(a*X-b)**2)	(-∞,∞)
Catenary	COSH(a*X)/(2*a)	(-∞,∞)

Table 2-1.b Cartesian Parametric forms. a,b,c,d = arbitrary constants.
C = COS($\bar{T}$), S = SIN($\bar{T}$), T = TAN($\bar{T}$)

Name:	X($\bar{T}$):	Y($\bar{T}$):	$\bar{T}$-range:
Cross-Curve	a/C	b/S	(-¶,¶) ≠0,±¶/2
Bullet-Nose	a*C	a/T	(-¶,¶) ≠0
Lemniscate of Bernoulli	a*C/(S**2+1)	a*S*C/(S**2+1)	[-¶,¶]
Lemniscate of Gerono	a*C	a*S*C	[-¶,¶]
Lissajous Patterns	a*S	b*SIN(c*$\bar{T}$+d)	
Eptrochoid	a*(b*C-c*COS(b*$\bar{T}$))	a*(b*S-c*SIN(b*$\bar{T}$))	
Hypotrochoid	a*(b*C+c*COS(b*$\bar{T}$))	a*(b*S-c*SIN(b*$\bar{T}$))	
Trochoid	a*($\bar{T}$-b*S)	a*(1-b*C)	(-∞,∞)
Pedals of a Parabola	a*C**2*(T**2-b)	a*S*C*(T**2-b)	(-¶/2, ¶/2)
Involute of a Circle	a*(C+$\bar{T}$*S)	a*(S-$\bar{T}$*C)	(-∞,∞)

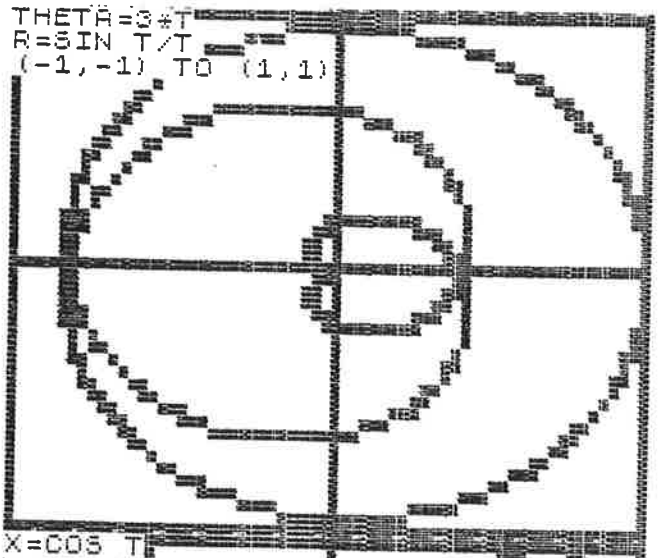
Table 2-1.c Polar forms. a,b = arbitrary constants.
C = COS($\bar{T}$), S = SIN($\bar{T}$), T = TAN($\bar{T}$)

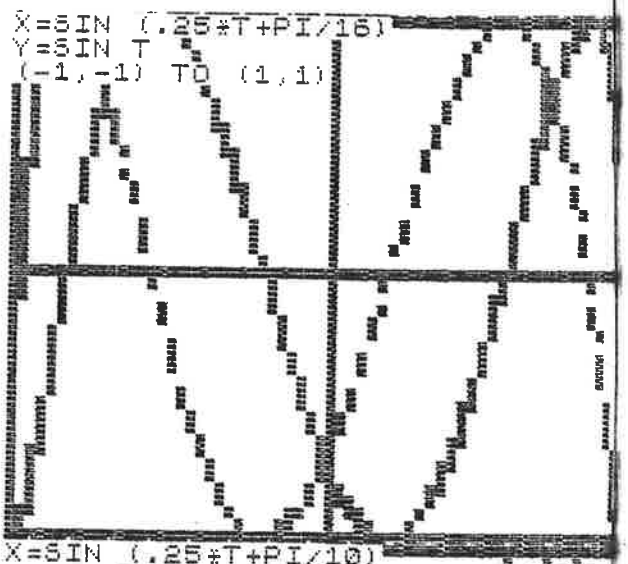
Name:	R($\bar{T}$):	$\bar{T}$-range:
Limaçon of Pascal	a*(C+b)	[-¶,¶]
Conchoid of Nichomedes	a*(C+b)/C	(-¶,¶) ≠±¶/2
Kappa Curve	a/T	(0,2*¶) ≠¶
Kampyle of Eudoxus	a/C**2	[-¶,¶] ≠±¶/2
Folia	a*C*(S**2-b)	
Folium of Descartes	a*C*S/(C**3+S**3)	
Semi-Cubical Parabola	a*T**2/C	
Cochleoid	a*S/$\bar{T}$	(-∞,∞) ≠0

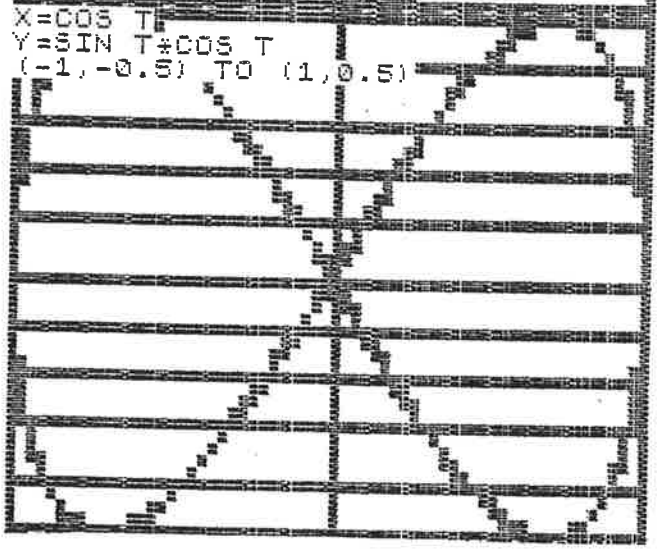
Table 2-1.d Polar Parametric forms. a,b = arbitrary constants.
C = COS(T), S = SIN(T).

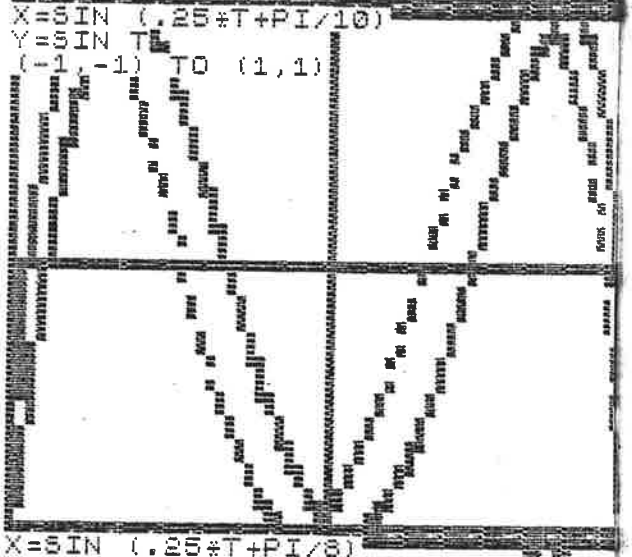
Name:	R(T):	T(T):	T-range
Rhodonea	a*C	T/b	
Nephroid of Freeth	a*(S+.5)	2*T	(-¶/2, ¶/2)
Cayley's Sextic	a*C**3	3*T	(-¶/2, ¶/2)
Tschirnhausen's Cubic	a/C**3	3*T	(-¶/2, ¶/2)
Logarithmic Spirals	a*EXP(T)	T/b	(-∞,∞)
Archimedes Spiral	a*T	T	(-∞,∞)
Hyperbolic Spiral	a/T	T	(-∞,∞) ≠0
Epi Spiral	a/C	T/b	(-¶/2, ¶/2)
Poinsot's Spiral #1	a/COSH(T)	T/b	(-∞,∞)
Poinsot's Spiral #2	a/SINH(T)	T/b	(-∞,∞) ≠0

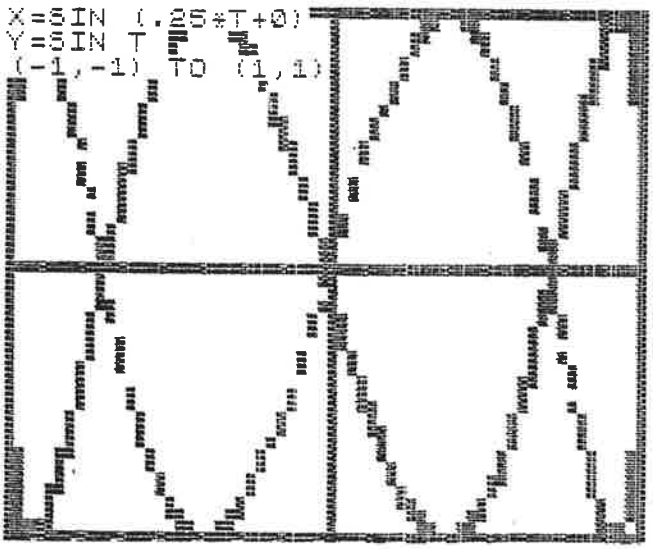
Figure 2-6 The following plots were made on the ZX81 and TS2068. The lower resolution plots were made on the ZX81, and the higher resolution plots were made on the TS2068 using a program similar to PLOT2.

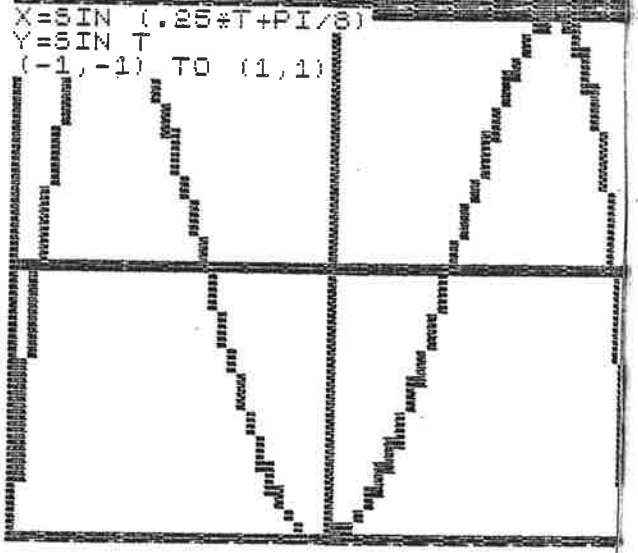

THETA=3*T
R=SIN T/T
(-1,-1) TO (1,1)

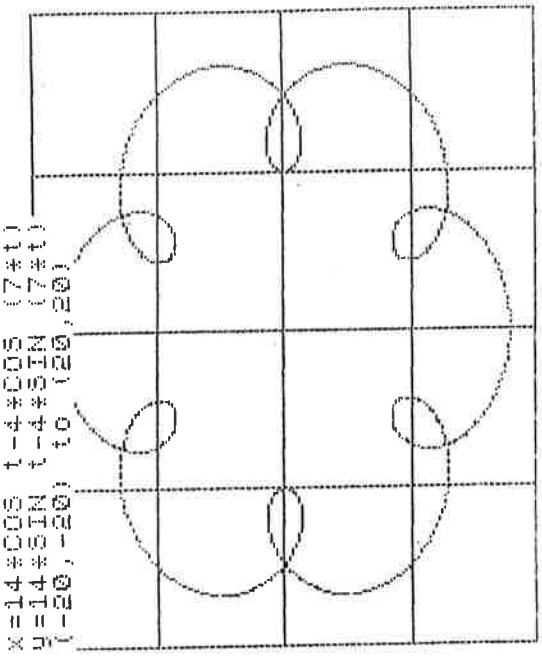

X=SIN (.25*T+PI/16)
Y=SIN T
(-1,-1) TO (1,1)

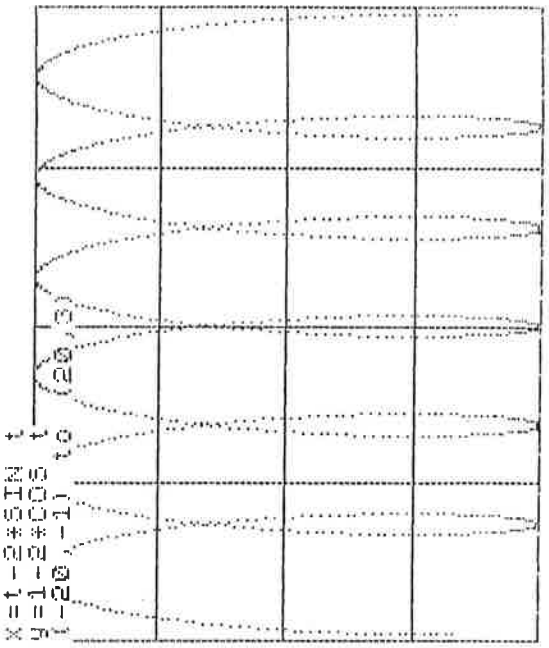

X=COS T
Y=SIN T*COS T
(-1,-0.5) TO (1,0.5)

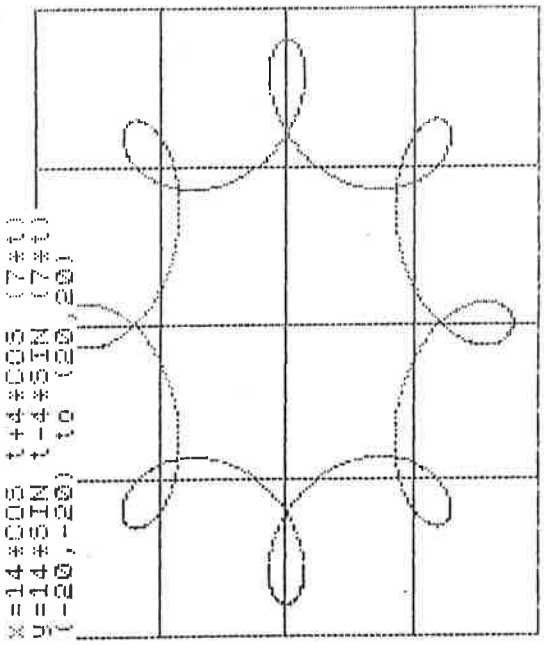

X=SIN (.25*T+PI/10)
Y=SIN T
(-1,-1) TO (1,1)

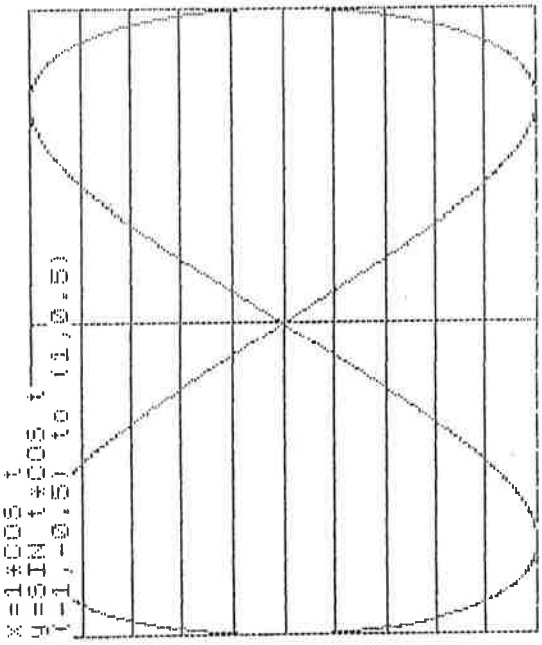

X=SIN (.25*T+0)
Y=SIN T
(-1,-1) TO (1,1)

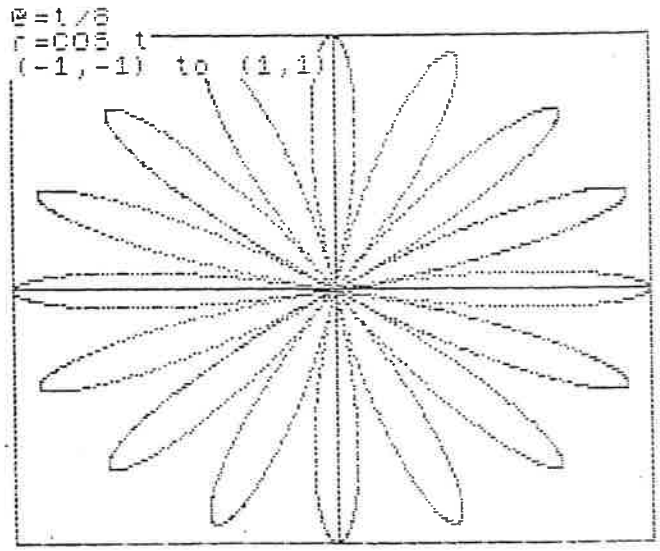

X=SIN (.25*T+PI/8)
Y=SIN T
(-1,-1) TO (1,1)

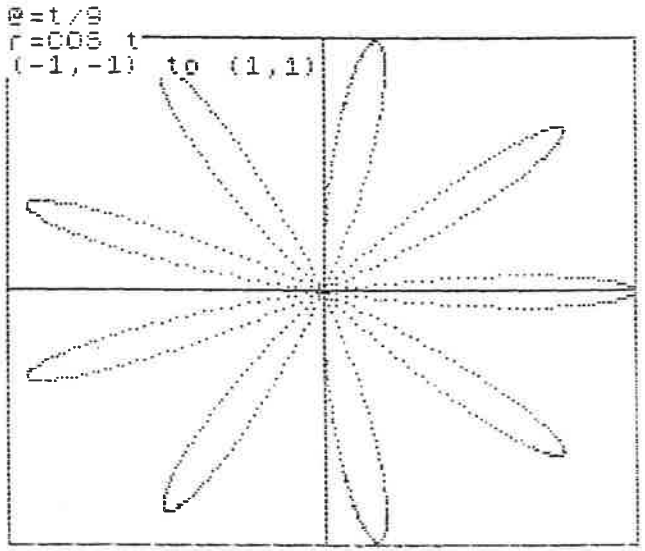

x=14*COS t-4*COS (7*t)
y=14*SIN t-4*SIN (7*t)
t -20,-20) to (20,20)

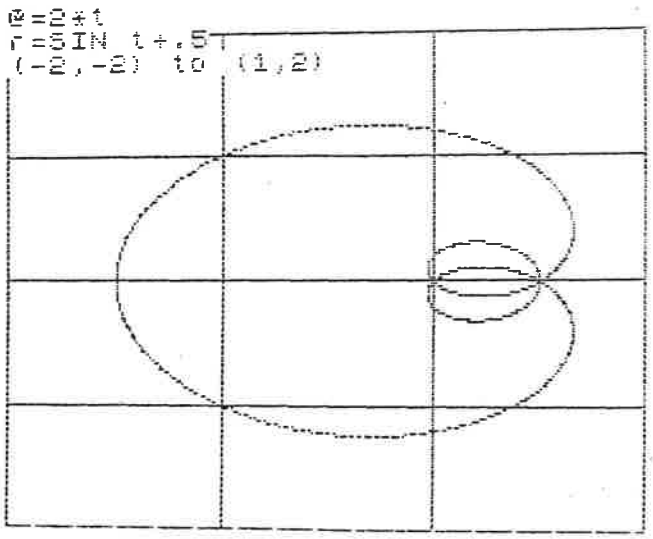

x=t-2*SIN t
y=1-2*COS t
t -20,-1) to (20,3)

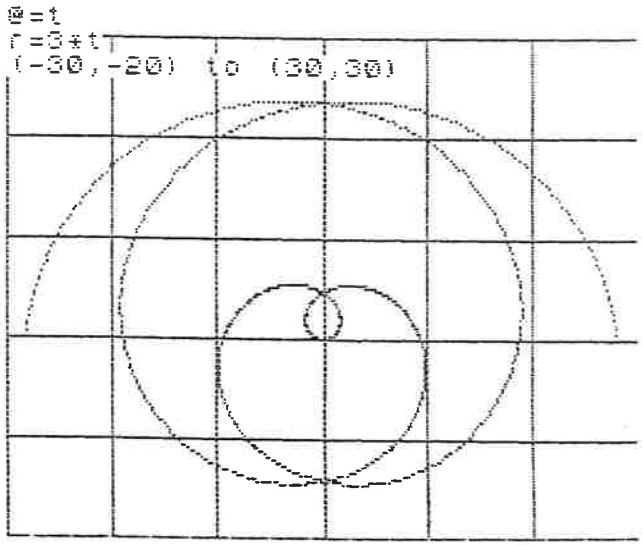

x=14*COS t+4*COS (7*t)
y=14*SIN t-4*SIN (7*t)
t -20,-20) to (20,20)


x=1*COS t
y=SIN t*COS t
t -1,-0.5) to (1,0.5)


@=t/8
r=COS t
(-1,-1) to (1,1)


@=t/9
r=COS t
(-1,-1) to (1,1)


@=2*t
r=SIN t+.5
(-2,-2) to (1,2)


@=t
r=3*t
(-30,-20) to (30,30)

```
2600 REM PROGRAM PLOT3 (PLOT CON
TOURS)
2602 PRINT "Z EXPRESSION?"
2604 PRINT "INPUT Z(X,Y)"
2606 INPUT Z$
2608 PRINT "CONTOUR-SPACING-PARA
METER?"
2610 PRINT "INPUT ORDER OF Z(X,Y
)"
2612 INPUT M
2614 PRINT "PLOTTING WINDOW?"
2616 PRINT "INPUT XMIN,XMAX,YMIN
,YMAX"
2618 INPUT A
2620 INPUT B
2622 INPUT C
2624 INPUT D
2626 LET E=(B-A)/255: REM *** FO
R ZX81, USE *** LET E=(B-A)/63
2628 LET F=(D-C)/175: REM *** FO
R ZX81, USE *** LET F=(D-C)/43
2630 LET M=10↑(2-M)/(255/2*E): R
EM *** FOR ZX81, USE *** LET M=1
0**(2-M)/(63/2*E)
2632 CLS
2634 FOR X=A TO B STEP E
2636 FOR Y=C TO D STEP F
2638 LET Z=INT (M*VAL Z$)
2640 IF INT (Z/2)*2=Z THEN PLOT
(X-A)/E,(Y-C)/F
2642 NEXT Y
2644 NEXT X
2646 PRINT AT 0,0;Z$;" UNITS/CON
TOUR= ";2/M
2648 PRINT "(";A;",";C;") TO (";
B;",";D;")"
2650 STOP
```

Figure 2-7 Contour plotting program

PLOT3

This program plots contours of 3-dimensional surfaces described by expressions of the form:

Z = F(X,Y)

The edges of the contours trace out plane curves in X and Y where Z has a particular value. The contours are spaced by the use of a Contour-Spacing-Parameter, C-S-P. For algebraic expressions, if the C-S-P equals the order of the expression (the highest degree in the expression), contours should be visible. Spacing may be determined, also, by the use of figure 2-8.

RUN:

PLOT3: Changes A B C D E F M X Y Z Z$
Prompts, in order of appearance:
1. "Z EXPRESSION?"
   "INPUT Z(X,Y)"
2. "CONTOUR-SPACING-PARAMETER?"
   "INPUT ORDER OF Z(X,Y)"
3. "PLOTTING WINDOW?"
   "INPUT XMIN,XMAX,YMIN,YMAX"

STOP: Contours plotted
Contour spacing and plotting window printed at top of screen.

Summary of operation:

Lines 2600 to 2624: Prompt user for expression, contour-spacing-parameter, and plotting window.
Lines 2626 to 2628: Compute step-sizes
Line 2630: Compute units per contour from C-S-P
Lines 2638 to 2640: Plot pixel if it lies on contour band.
Lines 2646 to 2648: Print units per contour and plotting window.

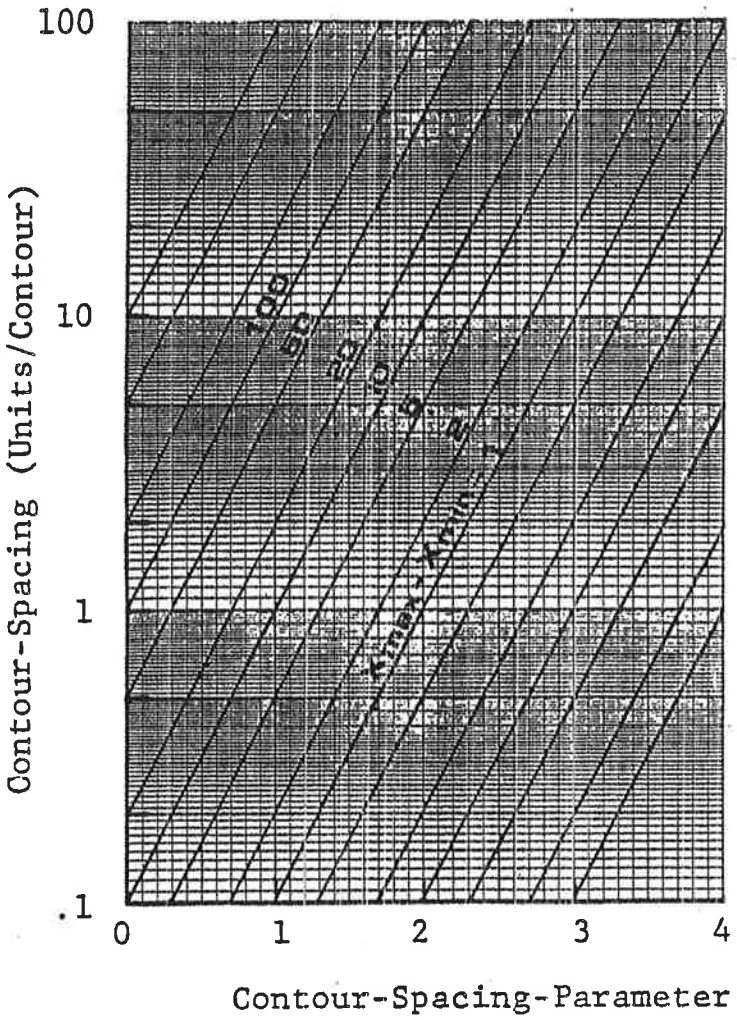


Figure 2-8 Contour spacing vs C-S-P and X-range for program PLOT3

Demonstration 2.5 PLOT CONTOURS OF SOME SURFACES FROM TABLE 2-2

Step 1: Enter program: PLOT3

Step 2 (PROBLEM 1): Plot contours of the elliptic paraboloid with equation

$$Z = X^2 + Y^2 + XY$$

Use the plotting window: (Xmin,Ymin) = (-10,-10)
(Xmax,Ymax) = (10,10)

Use a Contour-Spacing-Parameter of 2 since the highest order terms, X^2 and Y^2 have an exponent of 2: the expression is 2nd order. This gives 20 units per contour, (see figure 2-8).

Step 3: Run program:

USER ENTERS:	COMPUTER RESPONDS:
GOTO 2600	Z EXPRESSION? INPUT Z(X,Y)
X*X+Y*Y+X*Y	CONTOUR-SPACING-PARAMETER? INPUT ORDER OF Z(X,Y)
2	PLOTTING WINDOW? INPUT XMIN,XMAX,YMIN,YMAX
-10 10 -10 10	
{wait about 2m15s}	(results plotted)

Step 4 (PROBLEM 2): Plot contours of the elliptic cone with equation

$$Z = X^2 + Y^2$$

Use the plotting window: (Xmin,Ymin) = (-10,-10)
(Xmax,Ymax) = (10,10)

Use a Contour-Spacing-Parameter of 1, since the Z expression is first order. This gives 2 units per contour.

Step 5: Run program:

USER ENTERS:	COMPUTER RESPONDS:
GOTO 2600	Z EXPRESSION? INPUT Z(X,Y)
SQR (X*X+Y*Y)	CONTOUR-SPACING-PARAMETER? INPUT ORDER OF Z(X,Y)
1	PLOTTING WINDOW? INPUT XMIN,XMAX,YMIN,YMAX
-10 10 -10 10	
{wait about 6m30s}	(results plotted)

Step 6: Discussion:

The resulting plots have alternate zones of black and white, as shown in figure 2-9. The contours, loci of points having identical Z-values, are on the edges between these black and white zones.

In step 2, the elliptic paraboloid expression was second order. You might think of the variables X and Y as having units, say cm. Then, since $Z = X^2 + Y^2 + XY$, Z must have units of cm^2. How about expressions like

$$Z = aX + bX^2 + cX^3 \quad ?$$

In this case, Z is third order, taking the constant c to be dimensionless.

Selecting the Contour-Spacing-Parameter to be the order of the expression may or may not give the best plots. To change the spacing, use figure 2-8.

As you must have noticed, these plots take tremendous amounts of time. (As long as you aren't paying for computer time, who cares?) This is because each pixel of the screen must be examined to determine its status as either black or white (lines 2634 to 2644).

Step 7: Suggestions:

Try varying the Z expressions to give different patterns. To visualize the three-dimensional surfaces, label each contour edge, by determining the value of Z for one point and counting up using the contour spacing.

Plot the surfaces using different contour-spacing-parameters. In some cases, some interesting "interference" effects occur, especially when the C-S-P is small in relation to the order of the expression.

A faster algorithm, but more complicated one, is to start at a value of (X,Y) that gives a particular Z-value, and then tracing around the contour until the beginning of the contour, or the edge of the plotting window is reached.

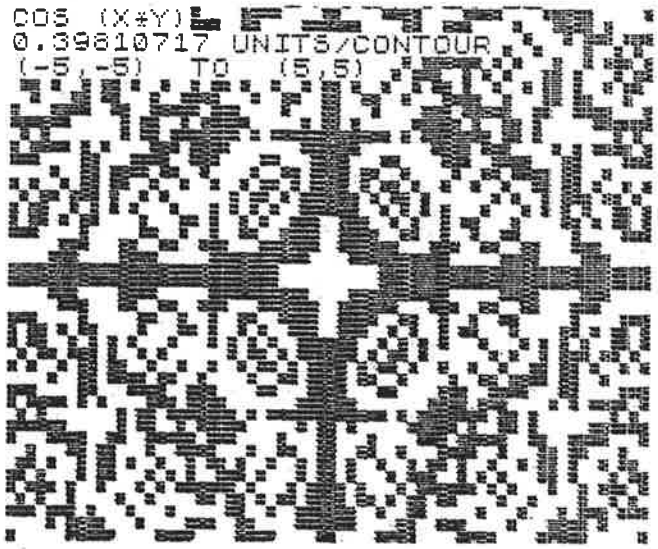
Table 2-2 Surfaces defined on X_Y plane. a,b,c = arbitrary constants.

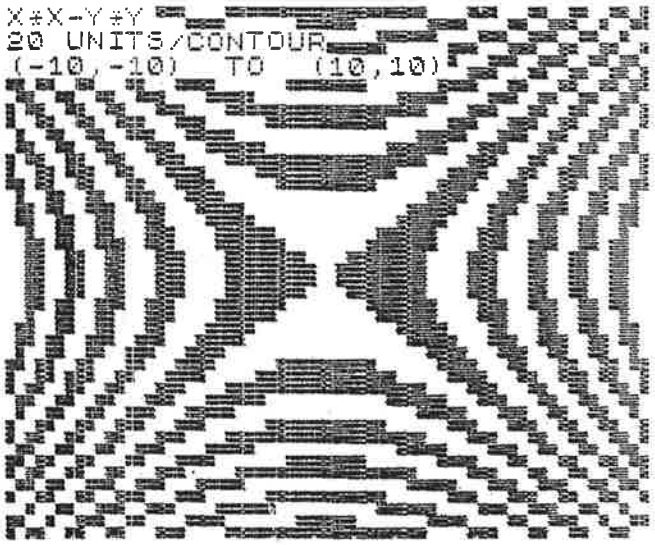
Name:	Z(X,Y):
Elliptic Cone	SQR((a*X)**2+(b*Y)**2)
Elliptic Paraboloid	(a*X)**2+(b*Y)**2+c*X*Y
Hyperbolic Paraboloid	(a*X)**2-(b*y)**2
|SIN(X+iY)|	SQR((COSH(2*Y)-COS(2*X))/2)
ARG(SIN(X+iY))	ATN(TANH(Y)/TAN(X))
Plane	a*X+b*Y+c
Family of Cassinian Ovals	(((X-a)**2+Y**2)*((X+a)**2+Y**2))**b
Family of Horse Fetters	(X**2+Y**2)/(4*a)+a*Y**2/(X**2+Y**2)

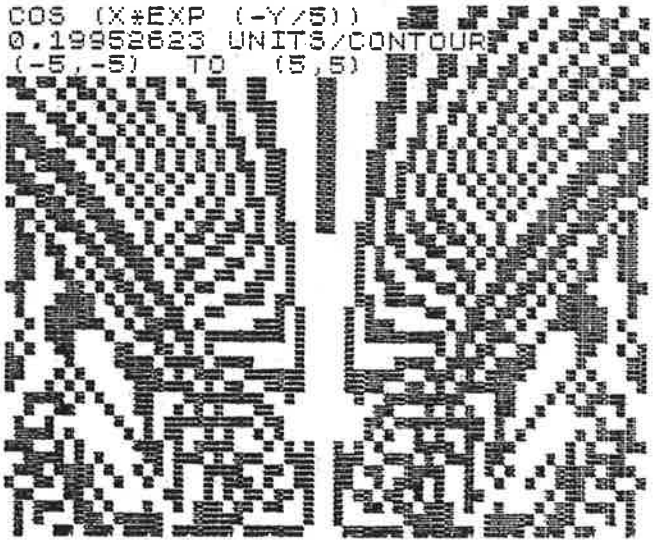
Examples:

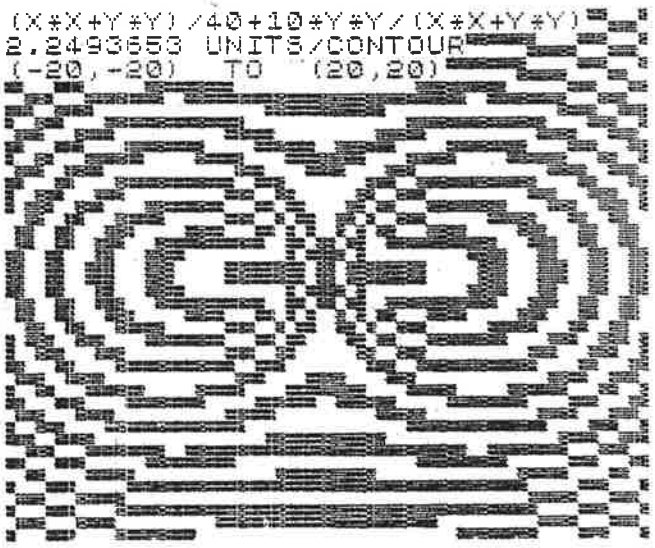
a:	b:	c:	C-S-P:	Window:	Time:
1	1		1	(-10,-10) (10,10)	6m30s
1	1	1	2	(-10,-10) (10,10)	2m15s
1	1		2	(-10,-10) (10,10)	2m15s
			.5	(-5,-5) (5,5)	13m
			.5	(-5,-5) (5,5)	12m15s
1	1	0	1	(-10,-10) (10,10)	1m45s
10	.25		1	(-20,-20) (20,20)	9m30s
10			.75	(-20,-20) (20,20)	3m15s

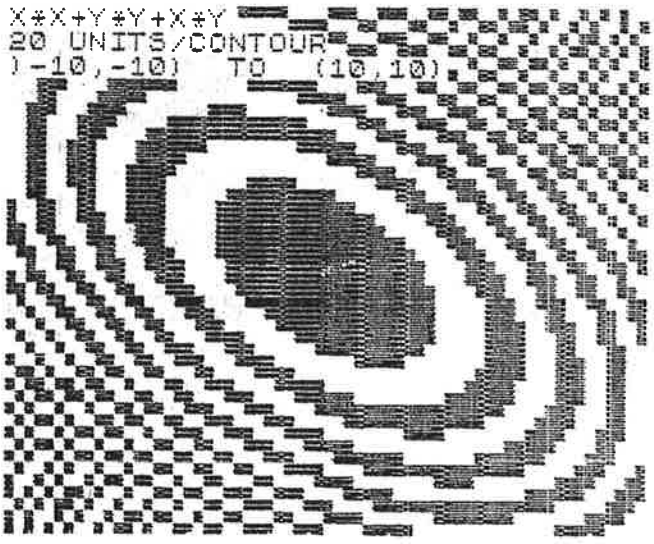
Figure 2-9 The following contour plots were made on the ZX81 and TS2068. Lower resolution plots were done on the ZX81 and higher resolution plots on the TS2068.

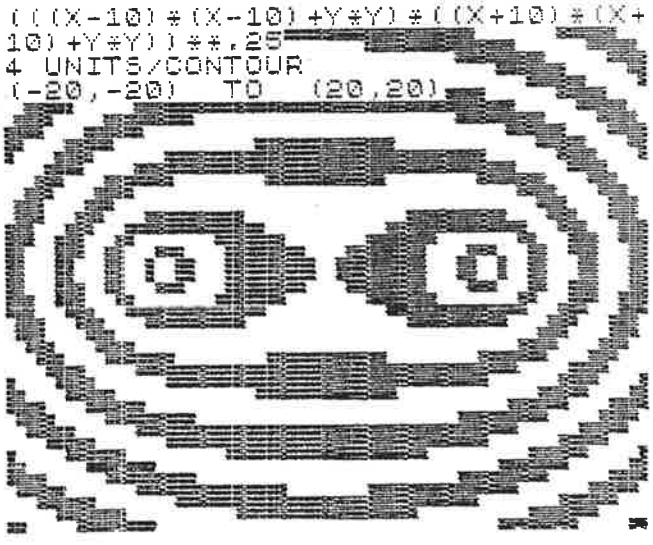


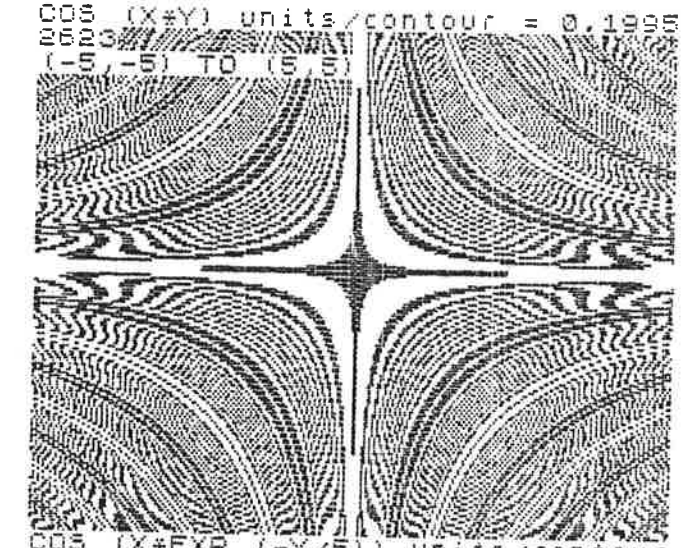


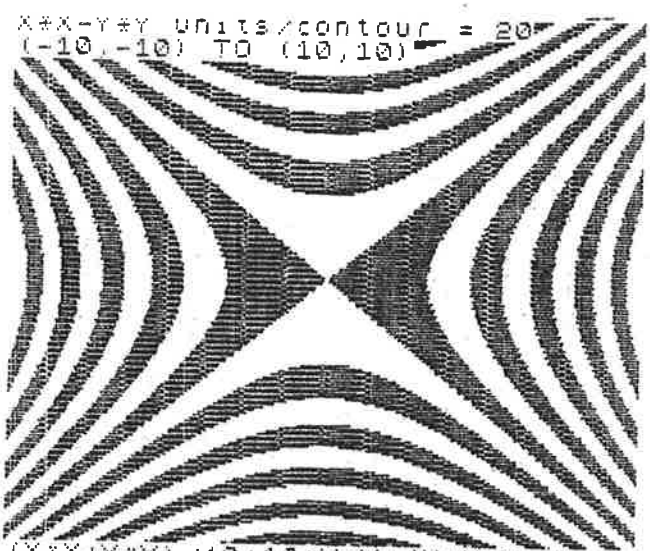


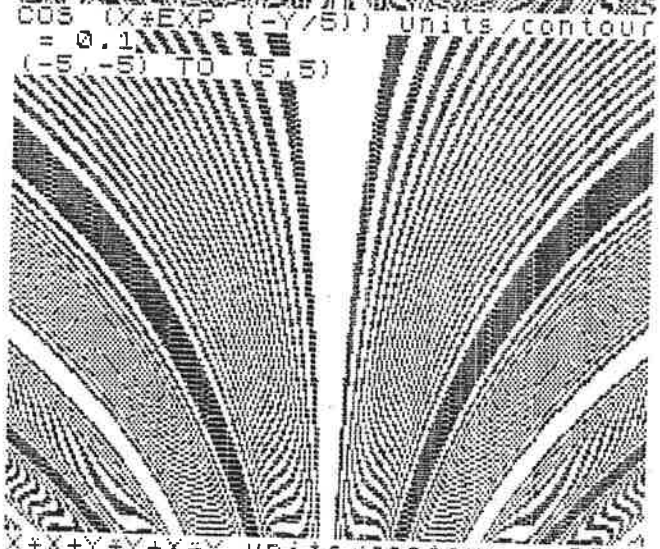


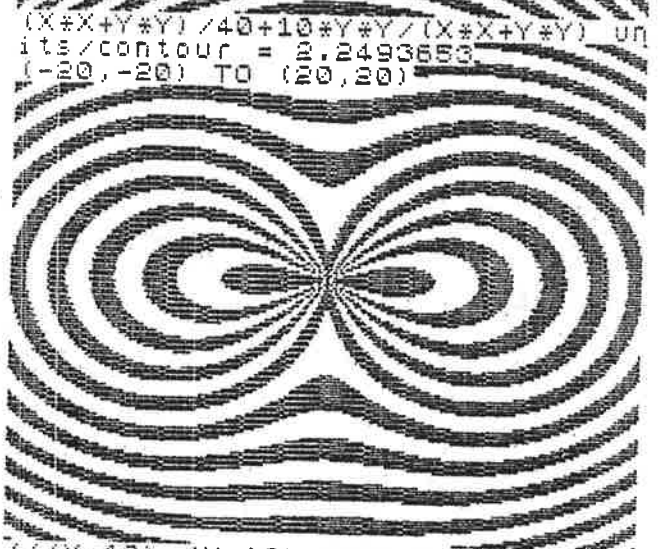


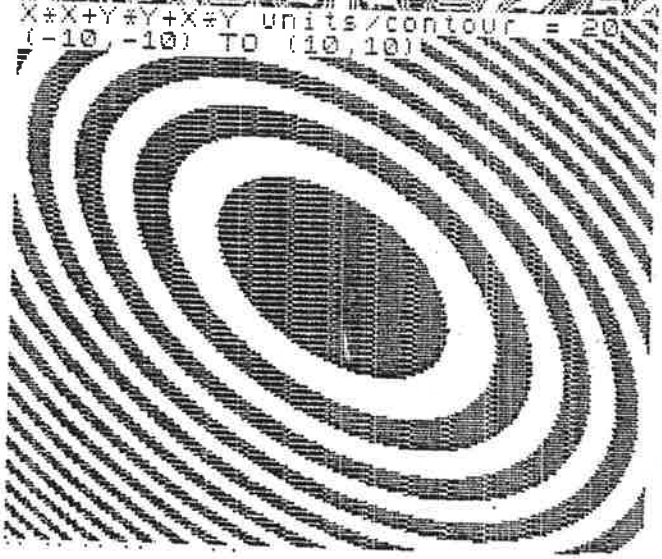


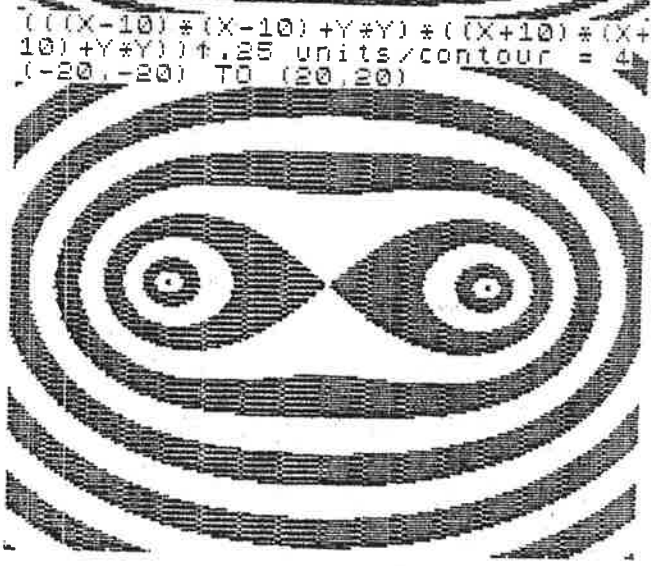

COS (X*Y) units/contour = 0.1995
2623
(-5,-5) TO (5,5)


X*X-Y*Y units/contour = 20
(-10,-10) TO (10,10)


COS (X*EXP (-Y/5)) units/contour
= 0.1
(-5,-5) TO (5,5)


(X*X+Y*Y)/40+10*Y*Y/(X*X+Y*Y) un
its/contour = 2.2493653
(-20,-20) TO (20,20)


X*X+Y*Y+X*Y units/contour = 20
(-10,-10) TO (10,10)


(((X-10)*(X-10)+Y*Y)*((X+10)*(X+
10)+Y*Y))↑.25 units/contour = 4
(-20,-20) TO (20,20)

Chapter 3 SPECIAL FUNCTIONS

There are many occasions when only complex answers to simple questions can be found. For example, here is a simple question: What is the circumference of a circle with a diameter of one? The solution, ¶, is a number which is irrational; that is, no ratio of integers will give its value exactly, (although the program RATIO, chapter 8, may be used to find an approximate ratio).

Fortunately for us all, the value of ¶ may be determined to any desired degree of accuracy by the use of any one of a number of techniques. The general idea is to add up a number of simple terms to obtain a complex approximation.

The three functions of this chapter are in the same boat with ¶ : they are all answers to fairly simple questions, but must be evaluated in complex ways.

The first function, erfc(x), is the solution of an "improper" integral (one with limits at +/- ∞):

$$\text{erfc}(x) = 2/\sqrt{\P}\int_x^{\infty} \text{EXP}(-t^2)dt \equiv \lim_{T\to\infty}\int_x^{T} 2*\text{EXP}(-t^2)/\sqrt{\P}\,dt$$

The integrand is two times the normal error function:

$$f(t) = \frac{1}{\sigma\sqrt{2\P}}\ \text{EXP}^{(-(t-m)^2/2\sigma^2)}$$

with m=0 and σ=.5

The subroutine ERFCX uses a series expansion to approximate values of the complementary error function. Demonstration 3.1 plots erfc(x) over 0<=X<=2.

The gamma function is the solution to another improper integral:

$$\Gamma(x) = \int_0^{\infty} t^{x-1}\text{EXP}(-t)dt \quad ; x>0$$

The definition is extended to negative non-integers by the recursion relation which the above formula obeys (see spec-sheet for GAMAX).

When x is an integer>0, then:

$$\Gamma(x) = (x-1)! \equiv (x-1)(x-2)\ \ldots\ *2*1$$

where 0! ≡ 1

The subroutine GAMAX evaluates $\Gamma(x)$ using both a series approximation and the recursion relation. Demonstration 3.2 plots the gamma function and demonstration 3.4 shows how GAMAX may be used to give values of factorials.

Finally, Bessel's functions are solutions to 2nd order differential equations of the form:

$$x^2\frac{d^2}{dx^2}y + x\frac{dy}{dx} + (x^2 - \nu^2)y = 0$$

The subroutine BESEX finds approximations to the Bessel's functions whose values are finite at x=0, (Bessel's functions of the first kind), and where ν in the above equation, (Bessel's equation), is a non-negative integer. Demonstration 3.3 plots $J_0(x)$.

There are many other examples of special functions. The sine and cosine functions are probably the most commonly encountered ones. sin(x) is the solution to:

$$\frac{d^2y}{dx^2} = -y \quad ; \quad y(0) = 0$$

Internal subroutines on the ZX81 and TS2068 evaluate sin(x) by the use of a series generator.

```
3000 REM ERFCX (COMPLEMENTARY ER
ROR FUNCTION)
3002 DIM W(6)
3004 LET W(6)=1
3006 LET W(5)=.0705230784
3008 LET W(4)=.0422820123
3010 LET W(3)=.0092705272
3012 LET W(2)=.0001520243
3014 LET W(1)=.0002765672
3016 LET V=.0000430638
3018 FOR I=1 TO 6
3020 LET V=V*X+W(I)
3022 NEXT I
3024 LET V=V↑-16
3026 RETURN
3100 REM GAMAX (GAMMA FUNCTION)
3102 DIM W(8)
3104 LET W(8)=1
3106 LET W(7)=-.577191652
3108 LET W(6)=.988205891
3110 LET W(5)=-.897056937
3112 LET W(4)=.918206857
3114 LET W(3)=-.756704078
3116 LET W(2)=.482199394
3118 LET W(1)=-.193527818
3120 LET V=.035868343
3122 LET M=INT X
3124 FOR I=1 TO 8
3126 LET V=V*(X-M)+W(I)
3128 NEXT I
3130 IF X<1 THEN GO TO 3140
3132 FOR I=1 TO M-1
3134 LET V=V*(X-I)
3136 NEXT I
3138 RETURN
3140 FOR I=0 TO ABS M
3142 LET V=V/(X+I)
3144 NEXT I
3146 RETURN
3200 REM BESEX (INTEGER ORDER BE
SSEL FUNCTION OF 1ST KIND)
3202 LET U=-X*X/4
3204 LET M=1
3206 FOR I=1 TO N
3208 LET M=M/I
3210 NEXT I
3212 LET V=M
3214 FOR I=1 TO 100
3216 LET M=M*U/(I*(I+N))
3218 IF ABS M<1E-9 THEN GO TO 32
24
3220 LET V=V+M
3222 NEXT I
3224 LET V=V*(X/2)↑N
3226 RETURN
```

Figure 3-1 Special functions subroutines: complementary error function, gamma function, and Bessel's functions

ERFCX

This subroutine calculates values of the complementary error function:

$$\text{erfc}(X) = 2/\sqrt{\pi}\int_{X}^{\infty} \text{EXP}(-t^2)dt \quad ; \quad -\infty < X < \infty$$

CALL: X = Argument

ERFCX: Changes I V W()

RETURN: V = erfc(X)

Example call:

```
LET ERFCX=3000
LET X=.5
GOSUB ERFCX
PRINT "ERFC(";X;") = ";V
```

Algorithm (see reference [1]):

$$\text{erfc}(X) = \left(1 + \sum_{I=1}^{6} a_I X^I\right)^{-16} + \text{error}(x) \quad ; \quad |\text{error}(x)| <= 3\times10^{-7}$$

$a_1 = 0.0705239784$
$a_2 = 0.0422820123$
$a_3 = 0.0092705272$
$a_4 = 0.0001520143$
$a_5 = 0.0002765672$
$a_6 = 0.0000430638$

GAMAX

This subroutine calculates values of the gamma function:

$$\Gamma(X) = \int_0^{\infty} t^{(X-1)}\,EXP(-t)\,dt \quad ; \quad X>0$$

$$\Gamma(X) = \Gamma(X+1)/X \quad ; \quad -\infty<X<\infty \; ; \; X\neq \text{non-positive integer}$$

CALL: X = Argument

GAMAX: Changes I M V W()

RETURN: V = $\Gamma(X)$

Example call:

```
LET GAMAX=3100
LET X=.5
GOSUB GAMAX
PRINT "GAMMA(";X;") = ";V
```

Algorithm (see reference [1]):

1. Let x = fractional part of X
   Let m = integral part of X
2. $\Gamma(x+1) = 1 + \sum_{I=1}^{8} b_I x^I + error(x) \quad ; \quad |error(x)|<=3\times10^{-7}$

b_1 = -0.577191652 (see Appendix B, γ)
b_2 = 0.988205891
b_3 = -0.897056937
b_4 = 0.918206857
b_5 = -0.756704078
b_6 = 0.482199394
b_7 = -0.193527818
b_8 = 0.035868343

3. $$\Gamma(X) = \begin{cases} \Gamma(x+1)\prod_{I=1}^{m-1}(X-I) & ; \; 2<=X \\ \Gamma(x+1) & ; \; 1<=X<2 \\ \dfrac{\Gamma(x+1)}{\prod_{I=0}^{|m|}(X+I)} & ; \; X<1 \end{cases}$$

BESEX

This subroutine calculates values of integer-order Bessel's functions of the first kind (finite at 0):

$J_\nu(X)$ = solution of (Bessel's equation):

$$X^2\frac{d^2Y}{dX^2} + X\frac{dY}{dX} + (X^2 - \nu^2)Y = 0 \quad ; Y(X=0) \text{ finite}$$

where ν = integer >=0.

CALL: N = Bessel's function order (ν in the above equation)
X = Argument

BESEX: Changes I M U V

RETURN: V = $J_N(X)$

Example call:

```
LET BESEX=3200
LET N=0
LET X=.5
GOSUB BESEX
PRINT "J";N;"(";X;") = ";V
```

Algorithm (see references [1], [2], and [7]):

1. The series representation of $J_N(X)$ is:

$$J_N(X) = \sum_{I=0}^{\infty} \frac{(-1)^I}{I!(N+I)!}(X/2)^{2I+N}$$

2. For a range of values of X, depending on the value of N, the series will converge before terms of the series become on the order of the machine round-off error. For $J_0(X)$, the range of X values is about 0<=X<=15. For larger arguments of X, an asymptotic series can be used: see reference [1].
3. The convergence test used in BESEX is a check to see if the current term of the series is less than 10^{-9}

Demonstration 3.1 PLOT COMPLEMENTARY ERROR FUNCTION

Step 1: Enter main program:

```
280 REM DEMO3.1
281 LET ERFCX=3000
282 FOR X=0 TO 2 STEP .02
283 GO SUB ERFCX
284 PLOT X*120,U*160: REM *** F
OR ZX81, USE *** PLOT X*30,U*40
285 NEXT X
286 STOP
```

Step 2: Enter subroutines: ERFCX

Step 3: Run program:

USER ENTERS:	COMPUTER RESPONDS:
RUN {wait about 25s}	(results plotted)

Step 4: Discussion:

The normal error function is

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \text{EXP}(-(x-m)^2/2\,\sigma^2)$$

The function is shown in the figure below for m=0 and $\sigma^2=\frac{1}{2}$ The complementary error function, erfc(x), represents the shaded area in the figure. If f(x) is the probability distribution function of some random variable X, then erfc(x) is the probability that X won't take on values inside the range (-x,x). For example, suppose x_i is the value of a particular measurement of a film thickness minus an assumed mean thickness:

$$x_i = t_{meas_i} - t_{assumed}$$

If X has a distribution like that of figure 3-2, then the probability that x_i has a value either above x or below -x is given by:

$$\text{Prob}(|X| > |x|) = \text{erfc}(x)$$

Since a normal error function has area of 1, erfc(0) = 1. The demonstration generated a plot of erfc(x) for 0<=x<=2, and shows that erfc(x) slowly decreases from 1 to 0, as x increases from 0.

Step 5: Suggestions

The error function

erf(x) = 1 - erfc(x)

represents the unshaded area in figure 3-2. Modify the main program or subroutine to give values of erf(x) for a plot. Also try the function

F(x) = 1 - erfc(x)/2

which represents the area of the curve in figure 3-2 for $-\infty <= X <= X$.
A property of erfc(x) which you may verify by example is:

erfc(-x) = 2 - erfc(x)

Generate a plot for $-2 <= X <= 2$.

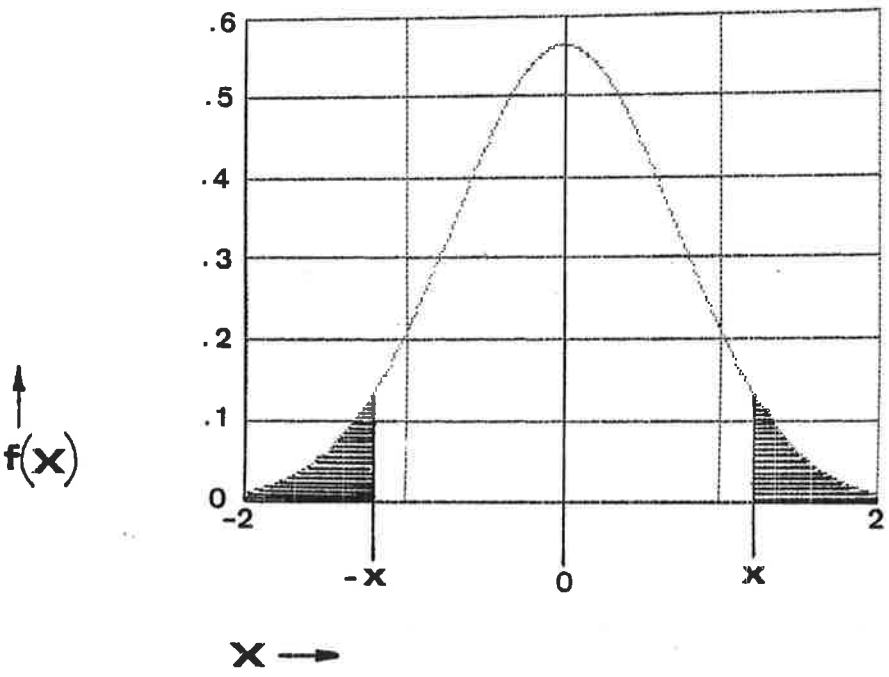


Figure 3-2 Normal error function

Demonstration 3.2 PLOT GAMMA FUNCTION

Step 1. Enter main program:

```
290 REM DEMO3.2
291 LET GAMAX=3100
292 FOR X=-3.9 TO 4 STEP .02
293 GO SUB GAMAX
294 IF V<-5 THEN LET V=-5
295 IF V>5 THEN LET V=5
296 PLOT (X+4)*30,(V+5)*16: REM
*** FOR ZX81, USE *** PLOT (X+4
)*7.5,(V+5)*4
297 NEXT X
298 STOP
```

Step 2: Enter subroutines: GAMAX

Step 3: Run program:

USER ENTERS:	COMPUTER RESPONDS:
RUN wait about 2m35s)	(results plotted)

Step 4: Discussion and suggestions:

Your plot should be similar to figure 3-3. The plot limits are the same for both plots. The gamma function is discontinuous at non-positive integers, as can be seen from figure 3-3. In fact, the function is undefined for non-positive integers.

Try different x-ranges for other plots: the resolution may improve for smaller windows.

For large values of x, Stirling's formula approximates $\Gamma(X)$:

$$\Gamma(X) \approx (X/e)^X \sqrt{2\pi/X}\left(1+1/12X\right) \quad ; \quad X \text{ "large"}$$

Modify the demonstration to compare this approximation with the values given by GAMAX. For another variation, modify GAMAX so that Stirling's formula is used if X is "large" enough. This will save time in the evaluation of $\Gamma(X)$, if X is very large.

Check to see if the subroutine GAMAX obeys the recursion relation given in the spec-sheet for GAMAX. List out a table of values of the function for arguments differing by integers.

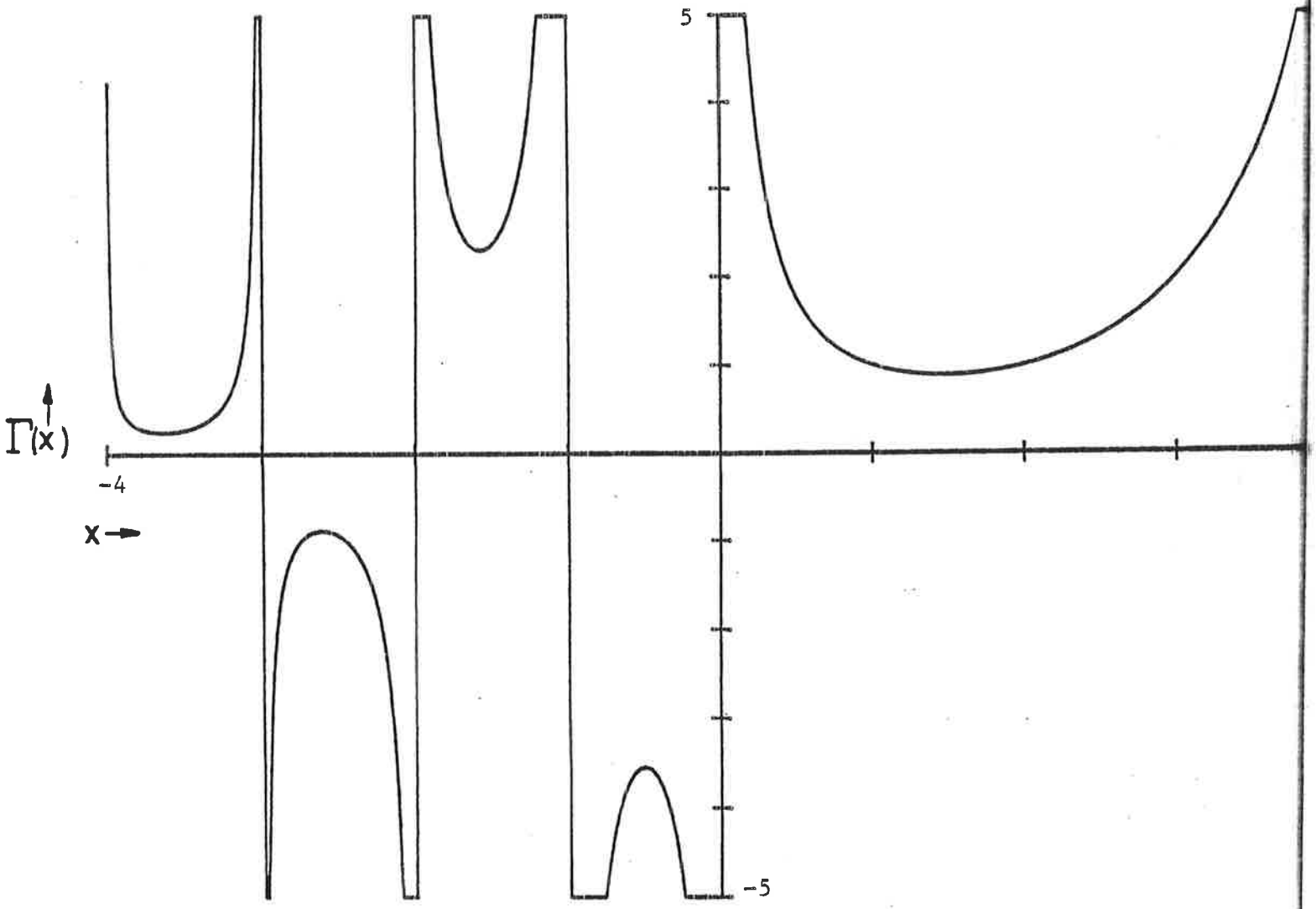


Figure 3-3 Gamma function

Demonstration 3.3 PLOT BESSEL'S FUNCTIONS

Step 1: Enter main program:

```
300 REM DEMO3.3
301 LET BESEX=3200
302 PRINT "INPUT NU=(INTEGER)=0
)"
303 INPUT N
304 CLS
305 FOR X=0 TO 10 STEP .1
306 GO SUB BESEX
307 PLOT X*24,(V+1)*80: REM ***
FOR ZX81, USE *** PLOT X*8,(V+1
)*20
308 NEXT X
309 STOP
```

Step 2: Enter subroutines: BESEX

Step 3 (PROBLEM): Plot $J_0(X)$, the 0^{th} order Bessel's function of the first kind, for $0<=X<=15$.

Step 4: Run program:

USER ENTERS:	COMPUTER RESPONDS:
RUN	INPUT NU=(INTEGER)=0)
0 {wait about 1m15s}	(results plotted)

Step 5: Discussion and suggestions:

The resulting plot should be similar to that of figure 3-4. The oscillatory behavior of the function brings to mind the sine and cosine functions. Bessel's functions are like two-dimensional analogues to the sine and cosine functions: A round drum head vibrates with a radial distortion function that is a Bessel's function or a sum of Bessel's functions. By contrast, a string vibrates with a distortion function of a sine function or a sum of sine functions.

The failure of BESEX at large arguments may be handled by an asymptotic series. This would increase the complexity of BESEX, but could reduce the evaluation times.

The relation between J_0 and J_1 is:

$$J_1(X) = -\frac{d}{dx} J_0(X)$$

In the next chapter, a subroutine to take derivatives is presented. Use this subroutine to check out the relationship between J_0 and J_1.

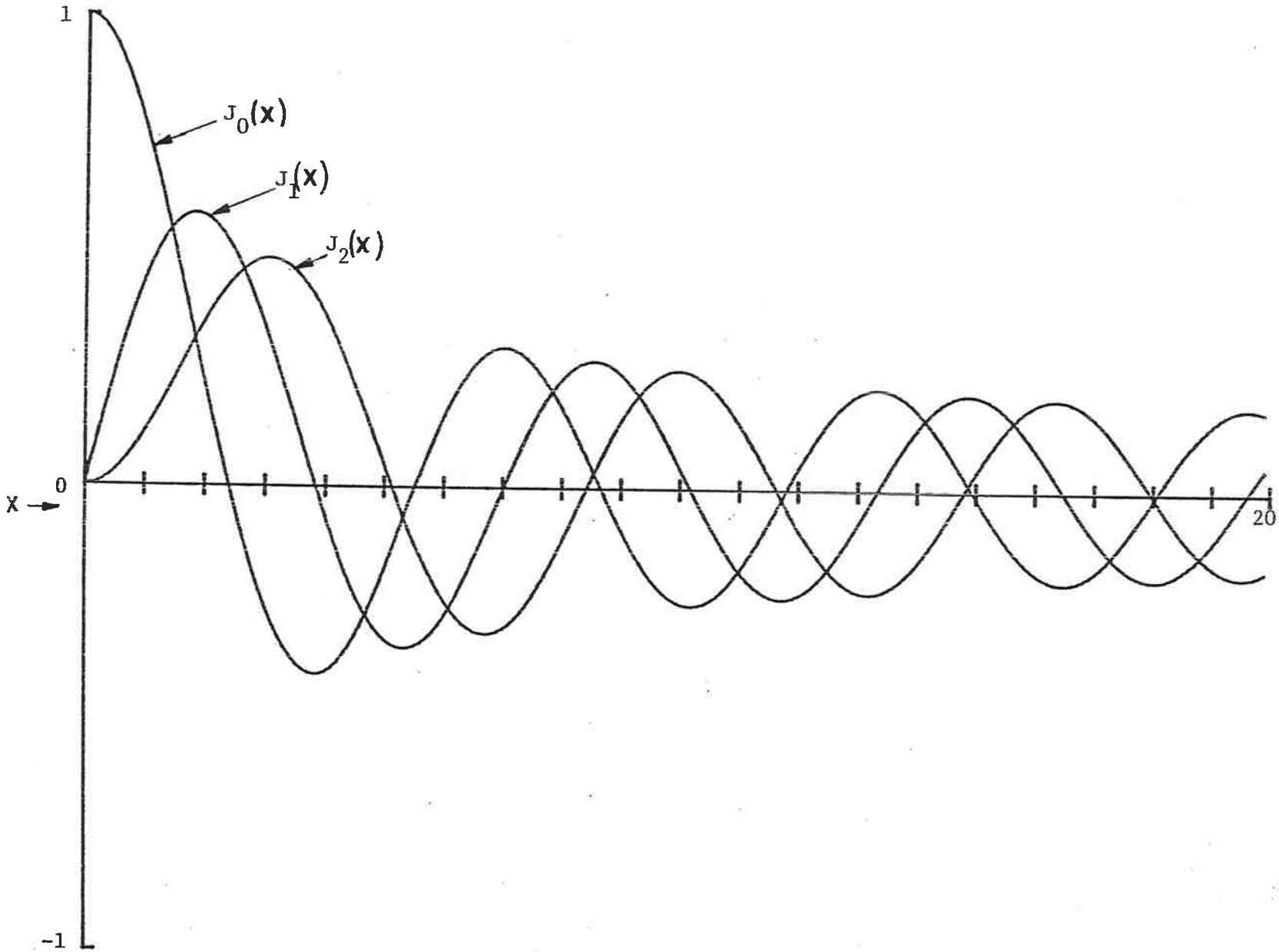


Figure 3-4 Bessel's functions of the first kind: J_0, J_1, and J_2

Demonstration 3.4 FACTORIALS

Step 1: Enter main program:

```
310 REM DEMO3.4
311 LET GAMAX=3100
312 PRINT "INPUT INTEGER>=0"
313 INPUT INTEGER
314 LET X=INTEGER+1
315 GO SUB GAMAX
316 CLS
317 PRINT INTEGER;"! = ";U
318 STOP
```

Step 2: Enter subroutines: GAMAX

Step 3 (PROBLEM): Calculate 15! = 15*14*13* ... *2*1

Step 4: Run program:

USER ENTERS:	COMPUTER RESPONDS:
RUN	INPUT INTEGER>=0
15	(results printed)

Step 5: Discussion and suggestions:

The gamma function is also called the factorial function since:

$n! = \Gamma(n+1)$; n = integer >=0

For this example, $15! \approx 1.3 \times 10^{12}$

Find the maximum n for which the demonstration program works. This may be fairly low. Use Stirling's formula, (see page 62), for higher values of n.

The gamma function may be used to evaluate non-integer order Bessel's functions:

$$J_\nu(X) = \sum_{I=0}^{\infty} \frac{(-1)^I}{I!\,\Gamma(\nu+I+1)} (X/2)^{2I+\nu}$$

Modify BESEX to take advantage of the above formula.

Chapter 4 CALCULUS

In Calculus, there are two operations performed on functions which are the inverse of each other. That is, one will "undo" the other (think of subtraction versus addition). These are integration and differentiation. To perform either one of these operations, the numerical approach is to sample the function at a finite number of closely-spaced points, and to assume that between each pair of points, the function can be well-approximated by an easily integrated or differentiated function.

The subroutine DERIV.1 performs a general-order differentiation of a sampled function. DERIV.1 assumes that the function is approximately linear between each pair of points. The first derivative of the function, at the point between a pair of sampled points, is given approximately by the slope of the line connecting the pair of points.

There is room left at line numbers 4100 to 4200 for a "DERIV.2" which was never written. This subroutine was to "symbolically" differentiate functions. That is, instead of chopping the function up and approximating the function by a line segment at each point, standard rules for differentiation would be used, such as

$$\frac{d}{dx}\,(x^n) = n\,x^{n-1}$$

The subroutine INTEG.1 integrates sampled functions. INTEG.1 uses a general Simpson's rule integration technique. Simpson's rule assumes that between pairs of points, the function can be approximated by a parabola, which is defined by three successive points. INTEG.1 may be used to integrate functions which have been sampled at uneven step-sizes. This feature is useful for the integration of functions which are rapidly changing in some regions, but slowly-changing in others. In addition, INTEG.1 calculates error estimates for both even and uneven step-size cases.

INTEG.2 integrates named functions by the use of a 10-point Gaussian quadrature technique. The subroutine samples the function ten times, at specific places, and assumes the function is well-approximated by a polynomial of degree 19 or less. INTEG.2 is, in general, faster than INTEG.1, however, no error estimate is calculated by INTEG.2, which may be necessary for many applications.

The demonstrations in this chapter illustrate the features of the three subroutines. For particular applications of the subroutines, some customization may be in order. For example, INTEG.1 calculates errors based on both even and uneven step-sizes. When only even step-sizes are used, some integration time may be saved by the elimination of appropriate program lines.

```
4000 REM DERIV.1 (GENERAL ORDER
DERIVATIVES OF SAMPLED FUNCTION)
4002 DIM D(G+1)
4004 FOR J=0 TO G
4006 LET D(J+1)=Y(J+I)
4008 NEXT J
4010 LET V=1
4012 FOR K=0 TO G-1
4014 LET V=V*(K+1)
4016 FOR J=1 TO G-K
4018 LET D(J)=(D(J+1)-D(J))/(X(I
+J+K)-X(I+J-1))
4020 NEXT J
4022 NEXT K
4024 LET V=V*D(1)
4026 RETURN
4200 REM INTEG.1 (GENERAL SIMPSO
N/3 INTEGRATION)
4202 LET Q=0
4204 LET R=0
4206 LET S=0
4208 LET T=0
4210 FOR I=1 TO N-2 STEP 2
4212 LET I1=X(I+1)-X(I)
4214 LET I2=X(I+2)-X(I)
4216 LET I3=I2-I1
4218 LET D1=(Y(I+1)-Y(I))/I1
4220 LET D2=(Y(I+2)-Y(I+1))/I3
4222 IF I>N-4 THEN GO TO 4236
4224 LET G=3
4226 GO SUB DERIV
4228 LET D3=V
4230 LET G=4
4232 GO SUB DERIV
4234 LET D4=V
4236 LET S=S+I2*(Y(I)+.5*D1*I2+(
2*I3-I1)*(D2-D1)/6)
4238 LET T=T+(I1*Y(I)+I2*Y(I+1)+
I3*Y(I+2))*.5
4240 LET R=R+D3*(I3-I1)*I2↑3/72
4242 LET Q=Q+D4*I1↑5/90
4244 NEXT I
4246 RETURN
4300 REM INTEG.2 (INTEGRATION; 1
0 PT. GAUSS)
4302 PRINT "INPUT INTEGRAND=F(X)
"
4304 INPUT Y$
4306 PRINT "INPUT LOWER,UPPER LI
MITS"
4308 INPUT A
4310 INPUT B
4312 GO SUB 4326
4314 PRINT "     ";B
4316 PRINT "    ⌠"
4318 PRINT "    ⌡ dX.(";Y$;")  = ";S
4320 PRINT "    "
4322 PRINT "     ";A
4324 STOP
4326 REM GAUSS
4328 DIM X(5)
4330 DIM W(5)
4332 LET X(1)=.1488743390
4334 LET X(2)=.4333953941
4336 LET X(3)=.6794095683
4338 LET X(4)=.8650633667
4340 LET X(5)=.9739065285
4342 LET W(1)=.2955242247
4344 LET W(2)=.2692667193
4346 LET W(3)=.2190863625
4348 LET W(4)=.1494513492
4350 LET W(5)=.0666713443
4352 LET S=0
4354 FOR I=1 TO 5
4356 LET X=(A+B+(A-B)*X(I))/2
4358 LET S=S+W(I)*VAL Y$
4360 LET X=A+B-X
4362 LET S=S+W(I)*VAL Y$
4364 NEXT I
4366 LET S=(B-A)*S/2
4368 RETURN
```

Figure 4-1 Calculus subroutines: Differentiation and integration

DERIV.1

This subroutine calculates a general order derivative of a sampled function, or a set of data points {(X(I),Y(I)); I=1,2,3,...}. The values of X() are assumed to be ordered from smallest to largest or largest to smallest, and the derivative at one point, $\hat{X}$, is calculated for a given value of the index I:

$$\frac{d^G}{dx^G} Y(\hat{X})$$

where $\hat{X} = \frac{X(I+G) + X(I)}{2}$

and the sampled function is: $Y(I) = f(X(I))$, $I = 1,2,3,...$

$X(1)<X(2)<X(3)<...$

or

$X(1)>X(2)>X(3)>...$

CALL:
- X() = Arguments of function in ascending order, or x-coordinates of data points.
- Y() = Function values at corresponding X(I)'s, or y-coordinates of data points.
- G = Order of derivative to be calculated
- I = Index of point where derivative is to be calculated.

DERIV.1: Changes D() J K V

RETURN: $V = \frac{d^G}{dX^G} Y(\hat{X})$; $\hat{X} = \frac{X(I+G) + X(I)}{2}$

Example call:

```
LET DERIV=4000
LET G=2
DIM X(100)
DIM Y(100)
  .
  .
  .
LET I=79
GOSUB DERIV
PRINT G;"-ORDER DERIVATIVE OF Y AT ";X(I);" = ";V
```

Algorithm:

1. Define:

$$Y^{(G)}_{K,J} \equiv \frac{d^G}{dX^G}\, Y(\hat{X}_{K,J})$$

$$\hat{X}_{K,J} \equiv \frac{X(I+K) + X(I+J)}{2}$$

$$\triangle X_{K,J} \equiv X(I+K) - X(I+J)$$

$$\triangle Y_{K,J} \equiv Y(I+K) - Y(I+J)$$

2. The general-order derivative is given, in this notation:

$$Y^{(G)}_{G,0} \approx \frac{Y^{(G-1)}_{G,1} - Y^{(G-1)}_{G-1,0}}{\hat{X}_{G,1} - \hat{X}_{G-1,0}}$$

3. For approximately equal stepsizes over X(I) to X(I+G):

$$\hat{X}_{G,1} - \hat{X}_{G-1,0} \approx \frac{\triangle X_{G,0}}{G}$$

4. Continuing in this way:

$$Y^{(G)}_{G,0} \approx G! \left[\frac{\dfrac{\triangle Y_{G,G-1}}{\triangle X_{G,G-1}} \quad \cdots \quad \dfrac{\triangle Y_{1,0}}{\triangle X_{1,0}}}{\dfrac{\dfrac{\triangle X_{G,2}}{\triangle X_{G,1}} - \dfrac{\triangle X_{G-1,1}}{\triangle X_{G-1,0}}}{\triangle X_{G,0}} - \dfrac{\dfrac{\triangle X_{G-1,1}}{\triangle X_{G-1,0}} - \dfrac{\triangle X_{G-2,0}}{\triangle X_{G-1,0}}}{\triangle X_{G,0}}} \right]$$

INTEG.1

This subroutine integrates a sampled function, or a set of data points {(X(I),Y(I));I=1,2,3,...,N}. The values of X() are assumed to be ordered from smallest to largest. A generalized Simpson's rule summation is used to approximate the integral:

$$\int_{X(1)}^{X(N)} Y(X)dx$$

Error estimates are calculated for both the cases of equal step-size and unequal step-size sampling. The trapezoidal rule summation is also calculated for estimating errors in "noisy" data integrations.

CALL: DERIV = 4000
X() = Arguments of function in ascending order, or x-coordinates of data points.
Y() = Function values at corresponding X(I)'s, or y-coordinates of data points.
N = Number of data samples (must be odd)
= Dimension of X() and Y()

INTEG.1: Changes D() D1 D2 D3 D4 G I I1 I2 I3 J K Q R S T V

RETURN: S = Simpson's rule summation
T = Trapezoidal rule summation
Q = Equal step-size estimated error
R = Unequal step-size estimated error

Example call:

```
LET INTEG=4200
LET DERIV=4000
LET N=21
DIM X(N)
DIM Y(N)
GOSUB INTEG
PRINT "SIMPSON/S RULE SUM = ";S
PRINT "TRAPEZOIDAL RULE SUM = ";T
PRINT "ESTIMATED ERROR = ";Q+R
```

Algorithm (see reference [16]):

1. Define:

$$\Delta X_{1,0} \equiv X(I+1) - X(I)$$

$$\Delta X_{2,0} \equiv X(I+2) - X(I)$$

$$\Delta X_{2,1} \equiv X(I+2) - X(I+1)$$

$$Y^{(K)} \equiv \frac{d^K}{dX^K} Y(X(I))$$

2. For the interval $\Delta X_{2,0}$, a Taylor series about X(I) gives:

$$S(I) \equiv \int_{X(I)}^{X(I+2)} Y(X)dX = \int_0^{\Delta X_{2,0}} d\Delta X \left(Y^{(0)} + Y^{(1)}\Delta X + \frac{Y^{(2)}}{2!}\Delta X^2 + \ldots \right)$$

3. The generalized Simpson's rule is:

$$S(I) = Y^{(0)}\Delta X_{2,0} + Y^{(1)}\frac{\Delta X_{2,0}^2}{2!} + Y^{(2)}\frac{\Delta X_{2,0}^3}{3!}(2\Delta X_{2,1} - \Delta X_{1,0})$$

$$+ \text{error}(I)$$

$$\text{error}(I) = \underbrace{\frac{Y^{(3)}}{3!} * \frac{\Delta X_{2,0}^3}{4!}(2\Delta X_{2,1} - 2\Delta X_{1,0})}_{\text{Unequal step-size error}} + \underbrace{\frac{Y^{(4)}}{4!} * \frac{\Delta X_{2,0}^5}{5!}}_{\text{Equal step-size error}}$$

4. The integral is given by

$$\int_{X(1)}^{X(N)} Y(X)dX \approx \sum_I S(I) \quad ; \quad I = 1,3,5,\ldots,N-2$$

5. The estimated error is given by

$$\text{Error(est)} = \sum_I \text{error}(I) \quad ; \quad I = 1,3,5,\ldots,N-2$$

INTEG.2

This program integrates a named function over user specified limits. The program uses a 10-point Gaussian quadrature algorithm, which is accurate for integrands which may be closely approximated by a polynomial of degree 19 or less, over the specified interval.

RUN:

INTEG.2: Changes A B I S W() X() Y$
Prompts, in order of appearance:
1. "INTEGRAND=F(X)"
2. "INPUT LOWER,UPPER LIMITS"

STOP: Integral printed

Algorithm (see reference [1]):

$$\int_A^B Y(X)dX \approx \sum_{i=1}^{5} w_i \left[F(r_i) + F(-r_i) \right]$$

where

$$F(r) = (B-A)/2 * Y\left[(B-A)/2 * r + (B+A)/2 \right]$$

r_i = i^{th} positive zero of the Legendre polynomial $P_{10}(r)$

$$w_i = \frac{2}{(1 - r_i^2)} \left[P'_{10}(r_i) \right]^2$$

Demonstration 4.1 PLOT DERIVATIVE

Step 1 (PROBLEM): Plot the 6^{th} derivative of SIN(X).

Step 2: Enter main program:

```
 320 REM DEMO4.1
 321 LET DERIV=4000
 322 DIM X(100)
 323 DIM Y(100)
 324 FOR I=1 TO 100
 325 LET X(I)=.04*PI*I
 326 LET Y(I)=SIN X(I)
 327 NEXT I
 328 LET G=6
 329 FOR I=1 TO 100-G
 330 PLOT 20*X(I),80*(Y(I)+1): R
EM *** FOR ZX81, USE *** PLOT 5*
X(I),20*(Y(I)+1)
 331 GO SUB DERIV
 332 PLOT 20*X(I),80*(V+1): REM
*** FOR ZX81, USE *** PLOT 5*X(I
),20*(V+1)
 333 NEXT I
 334 STOP
```

Step 3: Enter subroutines: DERIV.1

Step 4: Run program:

USER ENTERS:	COMPUTER RESPONDS:
RUN {wait about 2m20s}	(results plotted)

Step 5: Discussion and Suggestions:

The 6^{th} derivative of SIN(X) is -SIN(X). The resulting plot should be of 4 cycles of the sine and its 6^{th} derivative: X varies from 0 to 8¶, (see lines 324 and 325). That is, the number of cycles is

$$n = \frac{(Xmax - Xmin)}{2¶} = 4$$

The plot of the derivative is slightly out of phase with the original SIN(X). This is because the demonstration assumes that the calculated derivative is associated with the point (X(I),Y(I)), but the subroutine assumes the derivative is associated with the point

$$(\hat{X}(I),Y(I)) \quad ; \quad \text{where } \hat{X}(I) = \frac{X(I+G) - X(I)}{2}$$

The difference between $\hat{X}(I)$ and X(I) is small, since the step-size is very small. Correct for this shift either by changing the main program or changing DERIV.1 itself. Write a more general main program to handle any order of derivative, and take care of the "shift" problem, which is a function of the order of derivative

being taken.

Try different order derivatives of the sine function. To do this, change line 328 to

```
328 INPUT "ORDER?",G
```

Do a number of higher order calculations and get some points for a time versus derivative order plot. Use some of the subroutines from chapter 2 to plot these points.

The number of cycles for sampling and plotting may be changed by altering program lines 325, 330, and 332. It is interesting to note the effect of choosing a sampling such that only a very few number of samples are chosen per cycle. Remember that DERIV.1 assumes that the function is well-approximated by a piece-wise linear function between sampled points.

Demonstration 4.2 PLOT FIRST DERIVATIVE

Step 1: Enter main program:

```
340 REM DEM04.2
341 LET DERIV=4000
342 LET POINT=2100
343 LET VINAX=1000
344 LET SCALE=2300
345 LET PLOTD=2400
346 GO SUB POINT
347 LET G=1
348 FOR I=1 TO N-G
349 GO SUB DERIV
350 LET Y(I)=V
351 NEXT I
352 GO SUB PLOTD
353 STOP
```

Step 2: Enter subroutines: DERIV.1 POINT.2 VINAX SCALE.2 PLOTD

Step 3 (PROBLEM): Plot the first derivative of cosh(X) for -2<=X<=2.

Step 4: Run program:

USER ENTERS:	COMPUTER RESPONDS:
RUN	INPUT NO. PTS.
100	INPUT Y EXPRESSION=Y(X)
(EXP X+EXP -X)/2	INPUT XMIN,XMAX
-2	
2	
{wait about 45s}	(results plotted)

Step 5: Discussion and suggestions:

The derivative of cosh(X) is sinh(X), shcwn in figure 4-2. In Appendix A, ("One-liners"), expressions for these functions are given, in terms of the built-in functions (ZX81 and TS2068):

sinh(X) = (EXP X - EXP -X))/2
cosh(X) = (EXP X + EXP -X))/2

cosh(X) is an even function, which is to say that

cosh(-X) = cosh(X)

while sinh(X) is an odd function:

sinh(-X) = -sinh(X).

The first derivative of a function is the slope of the tangents

to the curve at each point. From figure 4-2, the slopes of the tangents to cosh(X) for X<0 are negative; the slopes of the tangents to cosh(X) for X>0 are positive; and the slope of the tangent to cosh(X) at X=0 is 0. The sinh function indeed shows these properties.

Some other functions are listed in table 2-1. Differentiate some of these functions by hand and by computer and compare your results.

Modify the demonstration to compile some data points for a plot of: difference between calculated derivative and actual derivative at each value of X. One might expect the error to be greatest for more rapidly-changing regions of the function. Vary also: the number of samples, and the function itself.

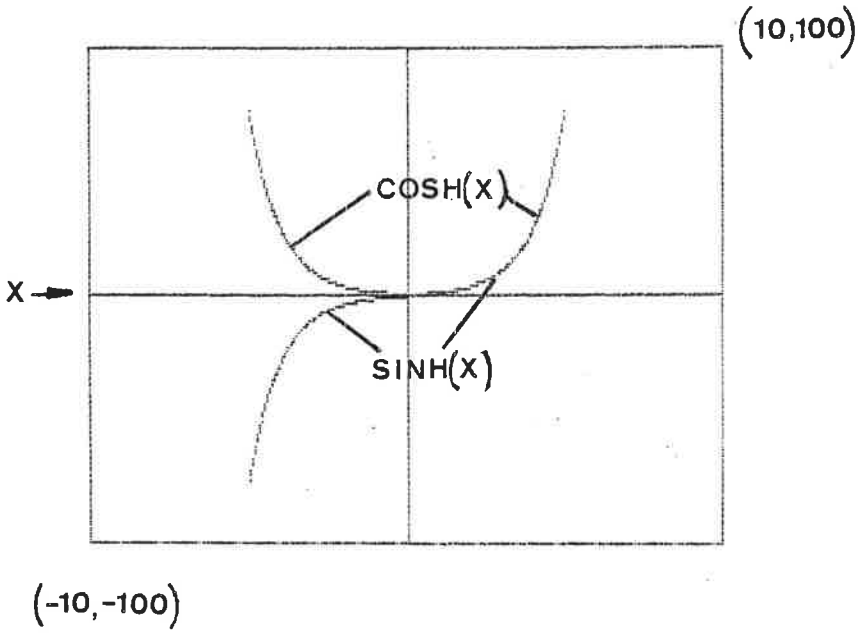


Figure 3-2 cosh(X) and sinh(X)

Demonstration 4.3 SIMPSON'S RULE INTEGRATION (EQUAL STEP-SIZE)

Step 1: Enter main program:

```
 360 REM DEMO4.3
 361 LET POINT=2100
 362 LET DERIV=4000
 363 LET INTEG=4200
 364 PRINT "NUMBER OF POINTS MUS
T BE ODD"
 365 GO SUB POINT
 366 GO SUB INTEG
 367 PRINT "INTEGRAL = ";S
 368 PRINT "ERROR(EST) = ";0
 369 STOP
```

Step 2: Enter subroutines: POINT.2 DERIV.1 INTEG.1

Step 3 (PROBLEM): Integrate SIN(X) for 0<=X<=¶

Step 4: Run program:

USER ENTERS:	COMPUTER RESPONDS:
RUN	NUMBER OF POINTS MUST BE ODD
	INPUT NO. PTS.
21	INPUT Y EXPRESSION=Y(X)
SIN X	INPUT XMIN,XMAX
0	
PI {wait about 25s}	(results printed)

Step 5: Discussion and suggestions

The result should be close to:

$$\int_0^{\pi} SIN(X)dX = -COS(\P) + COS(0) = -(-1) + (+1) = 2$$

This is the shaded area in figure 4-3. The ratio of area under the sine function to the area of the block in the figure is the "fill-ratio":

$$\frac{\text{Area under-sine}}{\text{Area under block}} = \frac{2}{\P} \doteq 64\%$$

The average value of the function between 0 and ¶ is

$$\frac{\text{Area under sine}}{\triangle X} = \frac{2}{\P} \doteq .64$$

Vary the number of sampled points and note the effect of this on the error calculations. Calculate the actual errors: between the hand-calculated value of the integral and the numerically inte-

grated values. Compare the actual error with the estimated error. Try some other functions and find out how well the error estimation works for each function. Make a plot of the estimated and actual errors versus number of sample points, for each function you investigate. Table 4-1 gives some results for the functions:

EXP(-X), SIN(X), $18x^5$ and $8/X^2$.

The error calculation used in this demonstration was for equal step-sizes. There is also an unequal step-size error calculation, used in demonstration 4.4. The unequal step-size errors should be small in this demonstration, since equal step-sizes were used. Print out this error calculation to see just how small they are.

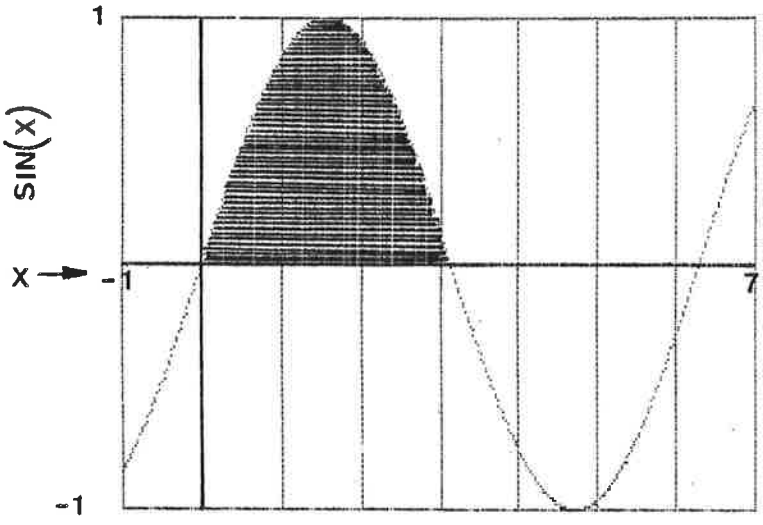


Figure 4-3 SIN(X)

Demonstration 4.4 SIMPSON'S RULE INTEGRATION (UNEQUAL STEP-SIZE)

Step 1 (PROBLEM): Integrate the function

$$Y = EXP(-X)$$

for X=0 to the end of 21 steps defined by

$$\triangle X_1 = .1$$
$$\triangle X_I = 1.2 * \triangle X_{I-1}$$

Step 2: Enter main program:

```
370 REM DEMO4.4
371 LET INTEG=4200
372 LET DERIV=4000
373 LET N=21
374 DIM X(N)
375 DIM Y(N)
376 LET X(1)=0
377 LET Y(1)=1
378 LET H=.1
379 FOR I=2 TO N
380 LET X(I)=X(I-1)+H
381 LET Y(I)=EXP -X(I)
382 LET H=H*1.2
383 NEXT I
384 GO SUB INTEG
385 PRINT "INTEGRAL = ";S
386 PRINT "ERROR(EST) =";R
387 STOP
```

Step 3: Enter subroutines: INTEG.1 DERIV.1

Step 4: Run program:

USER ENTERS:	COMPUTER RESPONDS:
RUN (wait about 30s)	(results printed)

Step 5: Discussion:

The unequal step-size sampling is shown in figure 4-4. More samples are taken in the region near X=0, where the function is more rapidly changing, as compared to the region for larger X. (Plot the function's derivative using DERIV.1).

If enough samples are taken, and from a small enough value of X to a large value of X, then the result of INTEG.1 should approximate the value of the (improper) integral:

$$\Gamma(1) = \int_0^\infty EXP(-X)dX = -EXP(-X) \Big|_0^\infty = 1$$

Since the sampling was done with unequal step-sizes, the demonstration uses the unequal step-size error calculation.

Step 6: Suggestions:

Try this experiment for various functions, including EXP(-X): For a range of values of p, sample a function with:

1. Equal step-sizes: $\triangle X_I = (.1)*p$

2. Unequal step-sizes: $\triangle X_I = (.1)*p^{(I-1)}$

To keep the range of integration, R=Xmax-Xmin, relatively constant, the number of samples, N, must change:

1. Equal step-sizes: $N_E = INT(10R/p)$

2. Unequal step-sizes: $N_U = INT(LN(10R(p-1)+1)/LN(p))$

Some values of N_E and N_U for R=20 are

p:	N_E:	N_U:
1.01	198	110
1.02	196	81
1.03	194	65
1.04	192	56
1.05	190	49
1.06	188	44
1.07	186	40
1.08	185	36
1.09	183	34
1.10	181	31

Now, gather some data for a plot of the absolute values of estimated and actual errors versus number of samples, N. Do this for both the equal and unequal step-size cases.

This will illustrate the improvement gained by the use of the unequal step-size sampling method.

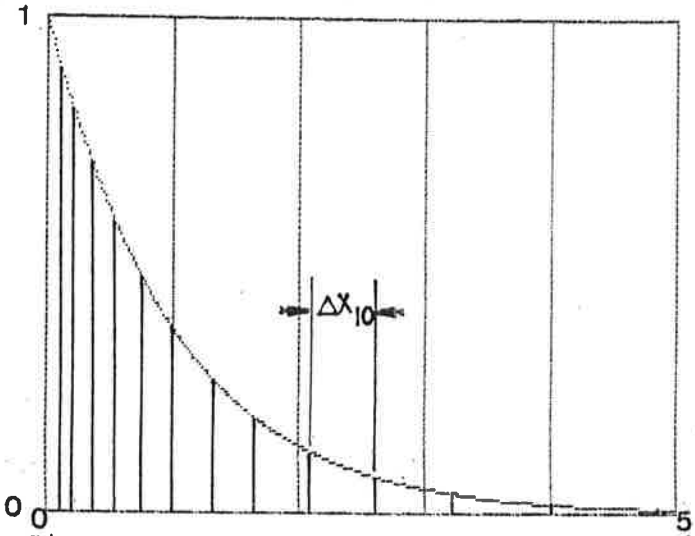


Figure 4-4 Unequal sampling of EXP(-X)

Demonstration 4.5 SIMPSON'S RULE INTEGRATION (ARTIFICIALLY NOISY DATA)

Step 1 (PROBLEM): Sample the function

$$Y(X) = EXP(-X)$$

with an equal step-size, for $0<=X<=1$. Then add "noise" to the sampled data and submit this noisy data to the integration subroutine INTEG.1. The "noise" in the example is .5% of the original sampled data at maximum.

Step 2: Enter main program:

```
390 REM DEM04.5
391 LET INTEG=4200
392 LET DERIV=4000
393 LET POINT=2100
394 PRINT "NUMBER OF POINTS MUS
T BE ODD"
395 GO SUB POINT
396 FOR I=1 TO N
397 LET NOISE=RND/500*SGN (RND-
.5)
398 LET Y(I)=Y(I)+NOISE
399 NEXT I
400 GO SUB INTEG
401 PRINT "INTEGRAL = ";S
402 PRINT "ERROR(EST) = ";ABS (
S-T)
403 STOP
```

Step 3: Enter subroutines: INTEG.1 DERIV.1 POINT.2

Step 4: Run program:

USER ENTERS:	COMPUTER RESPONDS:
RUN	NUMBER OF POINTS MUST BE ODD INPUT NO. PTS.
21	INPUT Y EXPRESSION=Y(X)
EXP -X	INPUT XMIN,XMAX
0 1	
{wait about 25s}(results printed)	

Step 5: Discussion and suggestions:

This demonstration presents an error estimation for integration of data corrupted by fairly large values of "noise". The "noise" in a real-world situation could be introduced by measurement inaccuracy, or fluctuation in the measured quantity. The error estimate in this demonstration is not a very good estimate of the actual error, but at least gives a "ball-park" number.

Try larger values of "noise" and different functions. Also try different "noise" calculations, by modifying line 397.

Table 4-1 Some results from integrations using INTEG.1. The functions were sampled N times over the interval (A,B). H is the step-size. Estimated error and actual errors are given as well as the ratio of the estimated to actual error.

F(X):	N:	A:	B:	H:	Time to Integrate:	Error (Actual):	Error (Estimated):	Ratio:
EXP(-X)	5	0	25	6.2	3s	1.1	.14	.125
	15			1.8	12s	.040	.0098	.242
	25			1.0	23s	.0058	.0023	.401
	75			0.34	72s	.000071	.000052	.723
SIN(X)	7	0	¶	0.52	5s	.00086	.0011	1.26
	21			0.16	19s	.0000068	.0000070	1.00
	35			0.09	33s	.00000081	.00000082	1.00
	105			0.03	100s	.000000009	.000000009	1.00
$18X^5$	9	0	1	0.12	7s	.0015	.0016	1.12
	27			0.04	25s	.000013	.000014	1.06
$8/X^2$	11	1	2	0.10	9s	.0001	.000065	.65
	33			0.03	31s	.00000098	.00000084	.86

Demonstration 4.6 GAUSSIAN INTEGRATION (POLYNOMIAL)

Step 1: Enter program: INTEG.2

Step 2 (PROBLEM): Integrate the function

$$F(X) = X^3 + X^2 + X + 1$$

for $0<=X<=10$.

Step 3: Run program:

USER ENTERS:	COMPUTER RESPONDS:
GOTO 4300	INPUT INTEGRAND=F(X)
X*X*X+X*X+X+1	INPUT LOWER,UPPER LIMITS
0	
10	(results printed)

Step 4: Discussion and suggestions:

The function

$$F(X) = X^3 + X^2 + X + 1 = (X + 1)(X^2 + 1)$$

is a polynomial of degree less than 19, so INTEG.2 gives an accurate result for its integration, which is:

$$\int_0^{10} F(X)dx = \left(\frac{X^4}{4} + \frac{X^3}{3} + \frac{X^2}{2} + X\right)\Big|_0^{10} = 2893.3\bar{3}$$

Integrate this function over different limits, and check the accuracy for different limits. Try higher degree polynomials. Use PLOT2, from chapter 2, for a clearer picture of the function and its integration.

Demonstration 4.7 GAUSSIAN INTEGRATION (NON-POLYNOMIAL)

Step 1: Enter program: INTEG.2

Step 2 (PROBLEM): See how well INTEG.2 integrates the function

$$Y(X) = (X-1)/LN(X)$$

for $0<=X<=1$.

Step 3: Run program:

USER ENTERS:	COMPUTER RESPONDS:
GOTO 4300	INPUT INTEGRAND=F(X)
(X-1)/LN X	INPUT LOWER,UPPER LIMITS
0	
1	(results printed)

Step 4: Discussion and suggestions:

Even though the function is not a polynomial, the result is close to the exact result:

$$\int_0^1 dX\ (X-1)/LN(X) \doteq .75$$

Apparently, the function is well approximated by a polynomial of degree 19 or less, at least in the interval $0<=X<=1$.

Try different limits and compare the results with hand-calculated values. Try integrating some of the functions in table 2-1.

Reference [1] gives different versions of the Gaussian quadrature technique. INTEG.2 may be improved if a higher version is used.

Chapter 5 MATRIX OPERATIONS

To see the forest despite the trees is a virtue well documented. For example, if we are given a set of equations to solve:

$$3x_1 + 2x_2 = 5$$

$$2x_3 + x_1 + x_2 = 7$$

$$x_2 + x_3 = 2$$

it is possible to think of the problem on a higher level: the transformation of a vector by an operator matrix:

$$\breve{A}\,\vec{x} = \vec{c}$$

where the matrix

$$\breve{A} = \begin{bmatrix} 3 & 2 & 0 \\ 1 & 1 & 2 \\ 0 & 1 & 1 \end{bmatrix}$$

and the vectors

$$\vec{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad \text{and} \qquad \vec{c} = \begin{bmatrix} 5 \\ 7 \\ 2 \end{bmatrix}$$

Matrices and vectors obey general rules, so that without looking too deeply into the details of the problem at hand, certain conclusions may be drawn. For example, if $\breve{A}$'s determinant is 0, the problem may have no unique solutions.

This higher-level approach to the problem is an example of "chunking", and its advantage in this particular case is that it gives a method whereby an infinite number of similar problems may be solved:

1. Call a subroutine to invert $\breve{A}$.
2. Call a subroutine to multiply $\vec{c}$ by $\breve{A}$'s inverse.

The two subroutines in this chapter accomplish these two steps. MXINV inverts a matrix, and MXMUT multiplies matrices. In both subroutines, the matrices to be multiplied or inverted are named in a string before calling the subroutine. This is the same approach as the vector operations subroutines.

Several bench-mark inversion times are given in FIG. 5-2. These are relatively long, and the reader may be able to find more efficient algorithms for matrix-inversion. It would also be a challenge to write the subroutine in assembly language, for much higher speeds.

```
5000 REM MXINV (MATRIX INVERSE,D
ETERMINANT)
5002 DIM U(N,2*N)
5004 DIM V(N,N)
5006 FOR I=1 TO N
5008 FOR J=1 TO N
5010 LET U(I,J)=VAL X$
5012 LET U(I,J+N)=0
5014 NEXT J
5016 LET U(I,I+N)=1
5018 NEXT I
5020 LET U=1
5022 FOR I=1 TO N
5024 LET L=0
5026 FOR J=I TO N
5028 IF ABS (U(J,I))<=L THEN GO
TO 5034
5030 LET L=U(J,I)
5032 LET K=J
5034 NEXT J
5036 LET U=U*L*(2*(K=I)-1)
5038 IF L<>0 THEN GO TO 5044
5040 PRINT "SINGULAR MATRIX"
5042 GO TO 5072
5044 FOR J=I TO 2*N
5046 LET M=U(I,J)
5048 LET U(I,J)=U(K,J)
5050 LET U(K,J)=M
5052 LET U(I,J)=U(I,J)/L
5054 NEXT J
5056 FOR J=1 TO N
5058 LET L=U(J,I)
5060 IF J=I OR L=0 THEN GO TO 50
68
5062 FOR K=I TO 2*N
5064 LET U(J,K)=U(J,K)-L*U(I,K)
5066 NEXT K
5068 NEXT J
5070 NEXT I
5072 FOR I=1 TO N
5074 FOR J=1 TO N
5076 LET V(I,J)=U(I,J+N)
5078 NEXT J
5080 NEXT I
5082 RETURN
5100 REM MXMUT (MATRIX-MULTIPLIC
ATION)
5102 DIM U(N,L)
5104 FOR I=1 TO N
5106 FOR K=1 TO L
5108 FOR J=1 TO M
5110 LET U(I,K)=U(I,K)+VAL X$
5112 NEXT J
5114 NEXT K
5116 NEXT I
5118 RETURN
```

Figure 5-1 Matrix operations subroutines: Matrix inversion and matrix multiplication

MXINV

This subroutine finds the inverse and determinant of a matrix named by the string X$. If the matrix is singular, (determinant = 0), an error message is printed.

CALL: N = Dimension of named matrix
X$ = Name of matrix, indexed by I and J

MXINV: Changes I J K L M U U(,) V(,)

RETURN: V(,) = Inverse of named matrix
U = Determinant of named matrix

Example call:

```
LET MXINV=5000
LET N=3
DIM A(N,N)
  .
  .
  .
LET X$="A(I,J)"
GOSUB MXINV
PRINT "DETERMINANT = ";U
```

Example of algorithm (Gaussian reduction, see reference [11]):

1. Invert the 2×2 matrix $\breve{A}$:

$$\breve{A} = \begin{bmatrix} 1 & 4 \\ 3 & 2 \end{bmatrix}$$

2. Lines 5000-5020: Load the left-hand side of the partitioned matrix $\breve{U}$ with $\breve{A}$ and the RHS of $\breve{U}$ with the 2×2 identity:

$$\breve{U} = \left[\begin{array}{cc:cc} 1 & 4 & 1 & 0 \\ 3 & 2 & 0 & 1 \end{array}\right]$$

Initialize the determinant variable:

U = 1

3. Lines 5024-5034: Find largest element in column 1 of $\breve{U}$, and its index, K:

K = 2
L = $\breve{U}$(K,1) = 3

4. Lines 5036-5042: Update determinant variable:

U = U*(-L) = -3

5. Lines 5044-5054: Exchange rows K and 1, then divide each element of the new first row by L:

$$\breve{U} = \left[\begin{array}{cc|cc} 1 & 2/3 & 0 & 1/3 \\ 1 & 4 & 1 & 0 \end{array}\right]$$

6. Lines 5056-5068: Subtract row 1 elements from row 2 elements, column by column:

$$\breve{U} = \left[\begin{array}{cc|cc} 1 & 2/3 & 0 & 1/3 \\ 0 & 10/3 & 1 & -1/3 \end{array}\right]$$

7. Lines 5024-5034: Find largest element in column 2 of $\breve{U}$, and its index, K:

$K = 2$
$L = \breve{U}(K,2) = 10/3$

8. Lines 5036-5042: Update determinant variable:

$U = U*L = -10$

9. Lines 5044-5054: Divide each element of the second row by L:

$$\breve{U} = \left[\begin{array}{cc|cc} 1 & 2/3 & 0 & 1/3 \\ 0 & 1 & 3/10 & -1/10 \end{array}\right]$$

10. Lines 5056-5068: Subtract 2/3 times row 2 elements from row 1 elements, column by column:

$$\breve{U} = \left[\begin{array}{cc|cc} 1 & 0 & -2/10 & 4/10 \\ 0 & 1 & 3/10 & -1/10 \end{array}\right]$$

11. Lines 5072-5080: Load $\breve{V}(,)$ with the RHS of $\breve{U}$:

$$\breve{V} = \left[\begin{array}{cc} -.2 & .4 \\ .3 & -.1 \end{array}\right]$$

12. The result is

$\breve{A}^{-1} = \breve{V}$
$\det(\breve{A}) = U$

MXMUT

This subroutine performs matrix multiplication between two matrices or vectors named in the string variable X$. The result is put into the matrix U(,).

CALL: X$ = String containing the names of two arrays or vectors to be multiplied:
1. Separated by a "*"
2. The left-hand-side matrix or vector is indexed by I,J or J
3. The right-hand-side matrix or vector is indexed J,K or J

N = Number of rows in left-hand-side array
M = Number of columns in left-hand-side array
= Number of rows in right-hand-side array
L = Number of columns in right-hand-side array

MXMUT: Changes I J K U(,)

RETURN: U(,) = result of matrix multiplication
U has dimensions U(N,L)

Example call:

```
LET MXMUT=5100
LET N=4
LET M=5
LET L=6
DIM A(N,M)
DIM B(M,L)
  :
  :
LET X$="A(I,J)*B(J,K)"
GOSUB MXMUT
```

Example of algorithm:

1. Calculate the product of matrices $\underset{\sim}{A}$ and $\underset{\sim}{B}$:

$$\underset{\sim}{U} = \underset{\sim}{A} * \underset{\sim}{B} = \begin{pmatrix} 1 & 2 \end{pmatrix} * \begin{bmatrix} 3 & 5 & 7 \\ 4 & 6 & 8 \end{bmatrix}$$

2. In this case:
   The number of rows in $\underset{\sim}{A}$: N=1
   The number of columns in $\underset{\sim}{A}$: M=2
   The number of rows in $\underset{\sim}{B}$:
   The number of columns in $\underset{\sim}{B}$: L=3

   The result matrix will have dimension U(1,3)

3. Each element of $\breve{U}$ is the Euclidean dot-product of the row vector of $\breve{A}$ with the column vector of $\breve{B}$, (see demonstration 1.2):

$$U(1,1) = ((1\ 2),(3\ 4)) = 1*3 + 2*4 = 11$$
$$U(1,2) = ((1\ 2),(5\ 6)) = 1*5 + 2*6 = 17$$
$$U(1,3) = ((1\ 2),(7\ 8)) = 1*7 + 2*8 = 23$$

4. The result is

$$\breve{U} = (11\ 17\ 23)$$

Demonstration 5.1 MATRIX INVERTER

Step 1: Enter main program:

```
 410 REM DEMO5.1
 411 LET MXINV=5000
 412 PRINT "INPUT DIM"
 413 INPUT N
 414 DIM A(N,N)
 415 PRINT "INPUT MX ELEMENTS:"
 416 FOR I=1 TO N
 417 FOR J=1 TO N
 418 REM *** FOR ZX81, USE *** S
CROLL
 419 PRINT "A(";I;",";J;")=? ";
 420 INPUT A(I,J)
 421 PRINT A(I,J)
 422 NEXT J
 423 REM *** FOR ZX81, USE *** S
CROLL
 424 PRINT "__________"
 425 NEXT I
 426 CLS
 427 LET X$="A(I,J)"
 428 GO SUB MXINV
 429 PRINT "DETERMINANT = ";U
 430 FOR I=1 TO N
 431 FOR J=1 TO N
 432 PRINT "A**-1(";I;",";J;") =
";V(I,J)
 433 LET M=J+(I-1)*N
 434 IF INT (M/15)*15<>M THEN GO
 TO 438
 435 PRINT "PRESS ""ENTER"" FOR
MORE"
 436 INPUT X$
 437 CLS
 438 NEXT J
 439 PRINT "__________"
 440 NEXT I
 441 STOP
```

Step 2: Enter subroutines: MXINV

Step 3 (PROBLEM 1): Find the inverse of the matrix

$$\overset{\smile}{A} = \begin{bmatrix} 1 & 2 \\ 0 & 1 \end{bmatrix}$$

Step 4: Run program:

USER ENTERS:	COMPUTER RESPONDS:
RUN	INPUT DIM
2	INPUT MX ELEMENTS: A(1,1)=?
1	A(1,2)=?
2	A(2,1)=?
0	A(2,2)=?
1	(inverse, determinant printed)

Step 5 (PROBLEM 2): Find the inverse of the matrix

$$\underset{\sim}{A} = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 0 & 1 & 2 & 3 & 4 \\ 0 & 0 & 1 & 2 & 3 \\ 0 & 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Step 6: Run program:

USER ENTERS:	COMPUTER RESPONDS:
RUN	INPUT DIM
5	INPUT MX ELEMENTS:
	A(1,1)=?
1	A(1,2)=?

... enter matrix elements row by row ...

0	A(5,5)=?
1 (wait about 17s)	(results printed)
	PRESS "ENTER" FOR MORE
(enter)	
(wait about 4s)	(results printed)

Step 7: Discussion and suggestions:

The matrices in step 3 and step 5 were both invertible: neither one of their respective determinants was zero. The two matrices were constructed so that the diagonal elements {A(1,1),...,A(N,N)} were all 1 and the elements below the diagonal were all 0. This guarantees that the matrices can be inverted, since the determinant in these cases is the product of the diagonal elements, or 1. Similarly constructed matrices, with various dimensions were inverted, and the inversion times are plotted in figure 5-**2**. The time required for the inversion of a matrix follows the relationship

$$t \propto N^{2.07}$$

where N is the dimension of the matrix.

Check the results of this demonstration by multiplying the original matrices by their respective inverted matrices. Use the matrix-multiplication subroutine MXMUT. The multiplication should give the identity matrix of the same dimension as $\underset{\sim}{A}$:

$$\underset{\sim}{I} = \begin{bmatrix} 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & & & & \vdots \\ 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & \cdots & 0 & 1 \end{bmatrix}$$

Find the inverted matrices of some:

1. Symmetric matrices: A(I,J) = A(J,I)

2. Identity matrices: A(I,J) = 0 if I<>J
   1 if I=J

3. Diagonal matrices: A(I,J) = 0 if I<>J

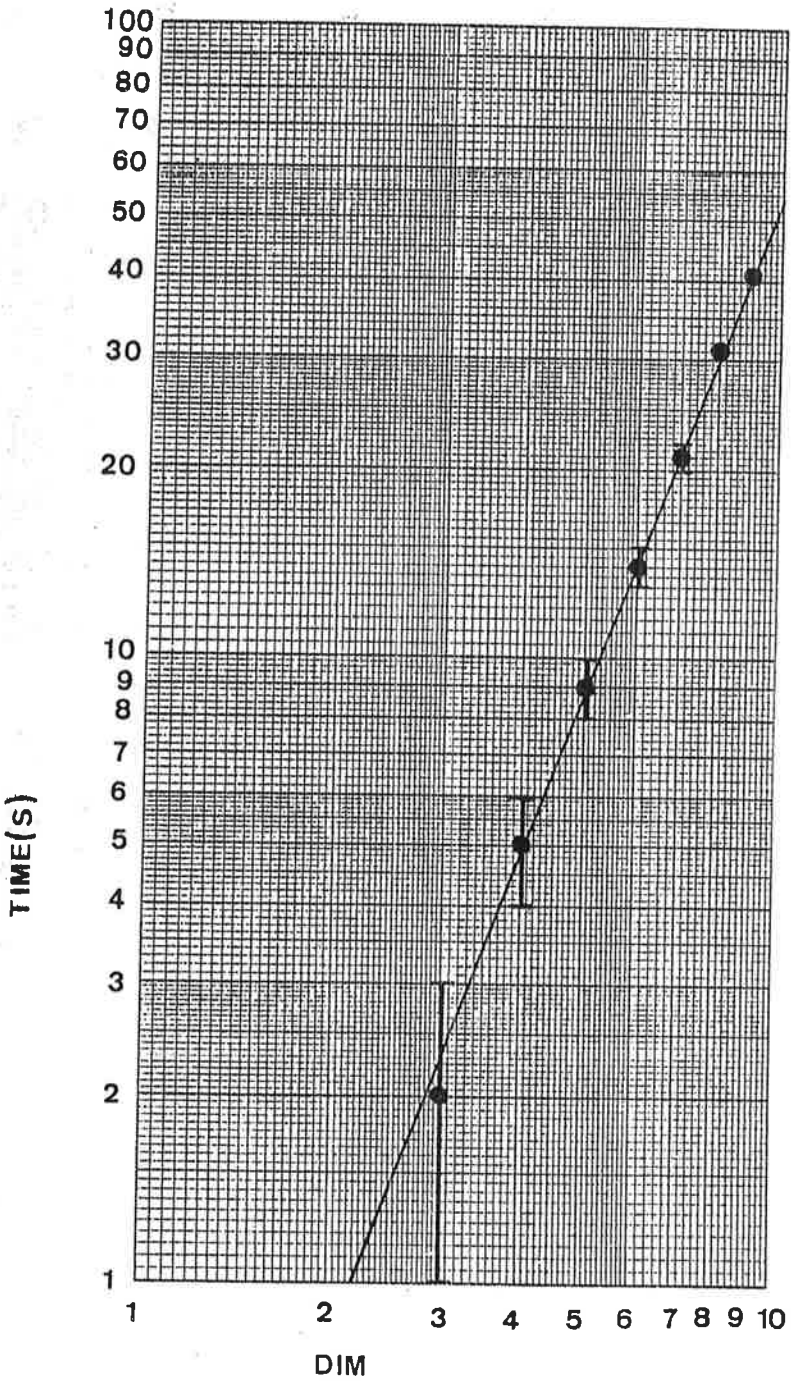


Figure 5-2 Inversion times vs dimension for matrices of same form as those of demonstration 5.1

Demonstration 5.2 MATRIX-MATRIX MULTIPLICATION

Step 1 (PROBLEM): Multiply matrix $\underset{\sim}{A}$ by matrix $\underset{\sim}{B}$:

$$\underset{\sim}{B}\,\underset{\sim}{A} = \begin{bmatrix} 0.6 & 1.0 & 1.4 & 1.8 & 2.2 \\ 0.8 & 1.2 & 1.6 & 2.0 & 2.4 \\ 1.0 & 1.4 & 1.8 & 2.2 & 2.6 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 2 & 1 \\ 3 & 1 \\ 4 & 1 \end{bmatrix}$$

Step 2: Enter main program:

```
450 REM DEMO5.2
451 LET MXMUT=5100
452 LET N=3
453 LET M=5
454 LET L=2
455 DIM B(N,M)
456 DIM A(M,L)
457 FOR J=1 TO M
458 FOR I=1 TO N
459 LET B(I,J)=I*.2+J*.4
460 NEXT I
461 LET A(J,1)=J-1
462 LET A(J,2)=1
463 NEXT J
464 LET X$="B(I,J)*A(J,K)"
465 GO SUB MXMUT
466 FOR I=1 TO N
467 FOR J=1 TO L
468 PRINT U(I,J)
469 NEXT J
470 PRINT "_____"
471 NEXT I
472 STOP
```

Step 3: Enter subroutines: MXMUT

Step 4: Run program:

USER ENTERS:	COMPUTER RESPONDS:
RUN	(results printed)

Demonstration 5.3 MATRIX-VECTOR MULTIPLICATION

Step 1 (PROBLEM): Multiply vector $\vec{X}$ by matrix $\overset{\smile}{B}$:

$$\overset{\smile}{B}\,\vec{X} = \begin{bmatrix} 0.6 & 1.0 & 1.4 & 1.8 & 2.2 \\ 0.8 & 1.2 & 1.6 & 2.0 & 2.4 \\ 1.0 & 1.4 & 1.8 & 2.2 & 2.6 \end{bmatrix} \begin{bmatrix} 2 \\ 4 \\ 6 \\ 8 \\ 10 \end{bmatrix}$$

Step 2: Enter program:

```
480 REM DEMO5.3
481 LET MXMUT=5100
482 LET N=3
483 LET M=5
484 LET L=1
485 DIM B(N,M)
486 DIM X(M)
487 FOR J=1 TO M
488 FOR I=1 TO N
489 LET B(I,J)=I*.2+J*.4
490 NEXT I
491 LET X(J)=J*2
492 NEXT J
493 LET X$="B(I,J)*X(J)"
494 GO SUB MXMUT
495 FOR I=1 TO N
496 PRINT U(I,1)
497 NEXT I
498 STOP
```

Step 3: Enter subroutines: MXMUT

Step 4: Run program:

USER ENTERS:	COMPUTER RESPONDS:
RUN	(results printed)

Demonstration 5.4 TRANSPOSED VECTOR-MATRIX MULTIPLICATION

Step 1 (PROBLEM): Multiply the matrix $\underset{\sim}{B}$ by the transposed vector $\vec{X}^T$.

$$\vec{X}^T \underset{\sim}{B} = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix} \begin{bmatrix} 0.6 & 1.0 & 1.4 & 1.8 & 2.2 \\ 0.8 & 1.2 & 1.6 & 2.0 & 2.4 \\ 1.0 & 1.4 & 1.8 & 2.2 & 2.6 \end{bmatrix}$$

Step 2: Enter main program:

```
500 REM DEMO5.4
501 LET MXMUT=5100
502 LET N=1
503 LET M=3
504 LET L=5
505 DIM B(M,L)
506 DIM X(M)
507 FOR I=1 TO M
508 FOR J=1 TO L
509 LET B(I,J)=I*.2+J*.4
510 NEXT J
511 LET X(I)=1
512 NEXT I
513 LET X$="X(J)*B(J,K)"
514 GO SUB MXMUT
515 FOR I=1 TO N
516 FOR J=1 TO L
517 PRINT U(I,J)
518 NEXT J
519 PRINT "_____"
520 NEXT I
521 STOP
```

Step 3: Enter subroutines: MXMUT

Step 4: Run program:

USER ENTERS:	COMPUTER RESPONDS:
RUN	(results printed)

Step 5: Discussion:

Demonstrations 5.2, 5.3, and 5.4 of the subroutine MXMUT show how any arbitrarily dimensioned matrix may be multiplied by another, as long as the number of columns of the left-hand-side matrix is equal to the number of rows of the right-hand-side matrix. The two subroutines, MXMUT and MXINV are used in the following two two chapters to solve a set of linear equations:

$$\underset{\sim}{A}\,\vec{X} = \vec{C}$$

where $\vec{X}$ is the solution vector, $\underset{\sim}{A}$ is a square coefficient-matrix, and $\vec{C}$ is a constant vector. The solution is

$$\vec{X} = \underset{\sim}{A}^{-1}\,\vec{C}$$

The implied multiplication is of a vector by a matrix, and $\underset{\sim}{A}^{-1}$ is the inverse of $\underset{\sim}{A}$.

For example, the following set of equations:

$\vec{X}^T$

$$X_1 + 2X_2 = 5$$

$$X_2 = 1$$

can be written in matrix form:

$$\begin{bmatrix} 1 & 2 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} 5 \\ 1 \end{bmatrix}$$

The solution is

$$\begin{bmatrix} X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 5 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & -2 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 5 \\ 1 \end{bmatrix} = \begin{bmatrix} 3 \\ 1 \end{bmatrix}$$

Step 6: Suggestions:

Demonstrations 5.2, 5.3, and 5.4 can be made more general by the addition of a matrix-element entry portion. This might be a separate subroutine, MXELM, which could also be called by any other program or subroutine to enter elements into arrays. Once you have this working, try multiplying several vectors, $\vec{X}$, by the matrix $\underset{\sim}{A}$:

$$\begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \end{bmatrix}$$

If you find a vector X which when multiplied by $\underset{\sim}{A}$ gives a multiple of itself:

$$\begin{bmatrix} 1 & 2 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} = K \begin{bmatrix} X_1 \\ X_2 \end{bmatrix}$$

then you have found an "eigen-vector" of the matrix $\underset{\sim}{A}$, and K is its corresponding "eigen-value."

Chapter 6 ROOTS/REGRESSION

This chapter contains three programs and subroutines to solve two common numerical analysis problems. The first problem is to find a solution of a set of equations which are not necessarily linear. The second is to find an equation to "fit" a set of data points. The three programs and subroutines, ROOTN, REGRP, and REGRN are included together in this chapter, because their structures are similar to one another, and they all make intrinsic use of the subroutines MXINV and MXMUT, from chapter 5.

The program ROOTN solves a general set of equations:

$$f_1(x_1,\ldots,x_N) = 0$$

$$\vdots$$

$$f_N(x_1,\ldots,x_N) = 0$$

The user must first supply an initial estimate of the solution vector $(x_1,\ldots,x_N)$, and the program iterates until a (not necessarily unique solution) is found.

The subroutines REGRP and REGRN find a set of parameters $\{w_J\}$ for an equation to fit a set of data points: $\{(x_1,\ldots,x_N,y)\}$. The resulting choice of parameters minimizes the error between the fitting-function and the data. In both subroutines, the form of the fitting equations must be known beforehand:

In REGRP: $f(\vec{x},\vec{w}) = w_1 f_1(x_1,\ldots,x_E) + \ldots + w_N f_N(x_1,\ldots,x_E)$ where the basis functions $f_1,\ldots,f_N$ are simply functions of $x_1,\ldots,x_E$, such as $\exp(x_1 + x_2)$.

In REGRN: $f(\vec{x},\vec{w})$ = any general function of $x_1,\ldots,x_E$ and parameters $\{w_J\}$.

REGRP calculates the parameters $\{w_J\}$ without any iteration, while REGRN requires both a user-supplied initial guess for the parameters, as well as several iterations to give the solution.

The reader and user of ROOTN, REGRP, and REGRN should have no trouble in finding real-world problems which can be solved by one or more of these programs and subroutines. This is their best test, and many improvements to the routines may be found in this way.

```
6000 REM PROGRAM ROOTN (NEWTON R
OOT-FINDER)
6002 LET MXINV=5000
6004 LET MXMUT=5100
6006 PRINT "INPUT NO.EQNS./VARS.
"
6008 INPUT N
6010 DIM F$(N,50)
6012 DIM X(N)
6014 DIM F(N)
6016 DIM D(N)
6018 DIM A(N,N)
6020 FOR J=1 TO N
6022 PRINT "INPUT F";J;"(X(1),..
.,X(N))"
6024 INPUT F$(J)
6026 PRINT "INPUT INITIAL X(";J;
")"
6028 INPUT X(J)
6030 CLS
6032 NEXT J
6034 LET P=0
6036 PRINT "ITER. ";P;,,,
6038 PRINT "J:  X(J):";TAB 18;"F
J:"
6040 FOR J=1 TO N
6042 LET F(J)=VAL F$(J)
6044 LET D(J)=(X(J)+(ABS X(J)<.0
01))*.001
6046 PRINT J;TAB 3;X(J);TAB 18;F
(J)
6048 NEXT J
6050 PRINT "------------"
6052 PRINT "CONTINUE?"
6054 INPUT X$
6056 IF X$="N" THEN STOP
6058 CLS
6060 FOR J=1 TO N
6062 FOR K=1 TO N
6064 LET M=X(K)
6066 LET X(K)=M+D(K)
6068 LET A(J,K)=(VAL F$(J)-F(J))
/D(K)
6070 LET X(K)=M
6072 NEXT K
6074 NEXT J
6076 LET X$="A(I,J)"
6078 GO SUB MXINV
6080 LET M=N
6082 LET L=1
6084 LET X$="U(I,J)*F(J)"
6086 GO SUB MXMUT
6088 FOR J=1 TO N
6090 LET X(J)=X(J)-U(J,1)
6092 NEXT J
6094 LET P=P+1
6096 GO TO 6036
```

Figure 6-1 Root finding program

ROOTN

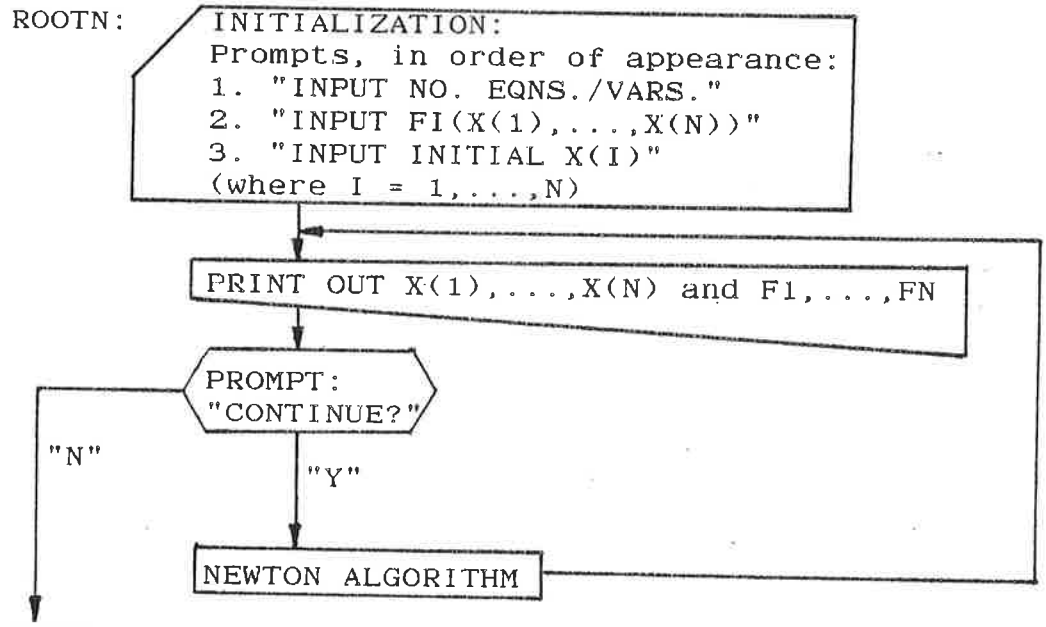
This program finds a solution vector, not necessarily unique, to a system of equations

$$F_1(X_1, X_2, \ldots, X_N) = 0$$
$$\vdots$$
$$F_N(X_1, X_2, \ldots, X_N) = 0$$

The program uses a general Newton's algorithm. The user supplies an initial guess for the solution vector, $(X_1, \ldots, X_N)$, and the program iterates to a solution. The number of equations is equal to the number if independent variables, $\{X_I\}$. At each iteration, the current value of the solution vector is printed. A root is found when the set of function values evaluated at the root is zero.

RUN:

ROOTN:



STOP:

Algorithm (General Newton):

Given a current (I^{th}) iteration estimate for the solution

$$\vec{X}_I = (X_{1_I}, \ldots, X_{N_I})$$

of the set of equations

$$F_1(\vec{X}) = 0$$
$$\vdots$$
$$F_N(\vec{X}) = 0$$

the new estimate, $\vec{X}_{I+1}$, is

$$\vec{X}_{I+1} = \vec{X}_I - \begin{bmatrix} \frac{\partial F_1(\vec{X}_I)}{\partial X_1} & \cdots & \frac{\partial F_N(\vec{X}_I)}{\partial X_1} \\ \vdots & & \vdots \\ \frac{\partial F_1(\vec{X}_N)}{\partial X_N} & \cdots & \frac{\partial F_N(\vec{X}_I)}{\partial X_N} \end{bmatrix}^{-1} \begin{bmatrix} F_1(\vec{X}_I) \\ \vdots \\ F_N(\vec{X}_I) \end{bmatrix}$$

Demonstration 6.1 SYSTEM OF LINEAR EQUATIONS

Step 1: Enter program and subroutines: ROOTN MXINV MXMUT

Step 2 (PROBLEM): Find the solution vector to the following system of linear equations

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 0 & 1 & 2 & 3 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X(1) \\ X(2) \\ X(3) \\ X(4) \end{bmatrix} = \begin{bmatrix} 30 \\ 16 \\ 7 \\ 2 \end{bmatrix}$$

Step 3: Run program:

USER ENTERS:	COMPUTER RESPONDS:
GOTO 6000	INPUT NO. EQNS./VARS.
4	INPUT F1(X(1),...,X(N))
X(1)+2*X(2)+3*X(3)+4*X(4)-30	INPUT INITIAL X(1)
1	INPUT F2(X(1),...,X(N))
X(2)+2*X(3)+3*X(4)-16	INPUT INITIAL X(2)
1	INPUT F3(X(1),...,X(N))
X(3)+2*X(4)-7	INPUT INITIAL X(3)
1	INPUT F4(X(1),...,X(N))
X(4)-2	INPUT INITIAL X(4)
1	(iteration 0 printed) CONTINUE?
{enter}	(iteration 1 printed) CONTINUE?
{enter}	(iteration 2 printed) CONTINUE?
N	(program stops)

Step 3: Discussion and suggestions:

The system of linear equations problem is easily solved by the Newton algorithm in one or two iterations. This is because the algorithm essentially linearizes the set of equations at each iteration. For example, when there is only one independent variable, a general problem might look like that of figure 6-2, where the root of F(X) is to be found. In the figure, the current estimate is X_I. The program finds the next iteration estimate, X_{I+1} by:

1. Find the tangent to F(X) at X_I.
2. X_{I+1} = X-intercept of the tangent line.

Find the number of iterations needed for various sizes of the coefficient matrix. Try some matrices that have elements with very large and very small values, to see if more than 2 iterations are required to find the solution.

Draw a picture of the general 2-dimensional root finding problem and compare it with the 1-d case shown in figure 6-2.

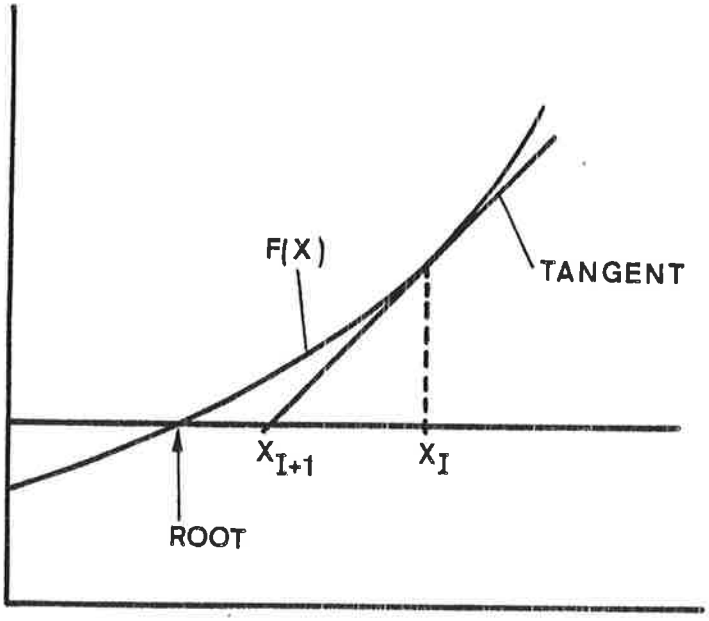


Figure 6-2 Root finding problem for one independent variable

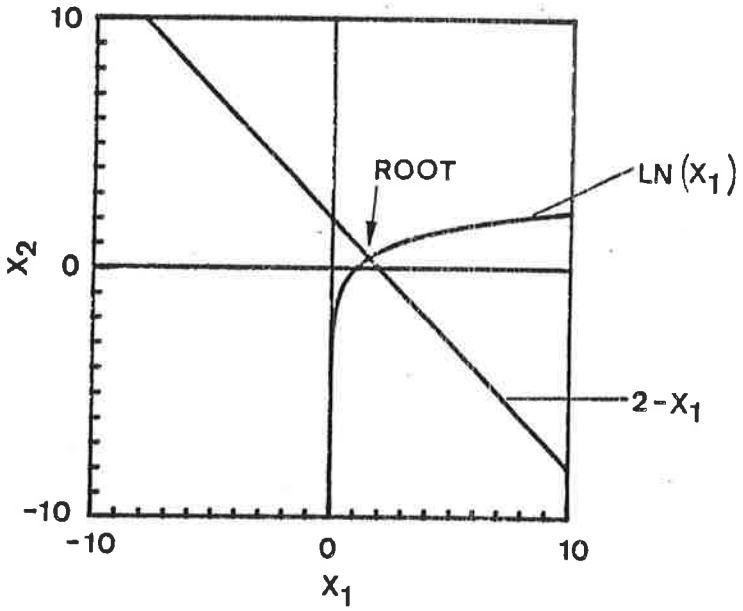


Figure 6-3 Root of demonstration 6.2

Demonstration 6.2 SYSTEM OF NON-LINEAR EQUATIONS

Step 1: Enter program and subroutines: ROOTN MXINV MXMUT

Step 2 (PROBLEM): Find a solution of the following system of non-linear equations:

$$\frac{EXP(X_2)}{X_1} = 1$$

$$X_1 + X_2 = 2$$

with initial guess: $X_1 = X_2 = 1$

Step 3: Run program:

USER ENTERS:	COMPUTER RESPONDS:
GOTO 6000	INPUT NO. EQNS./VARS.
2	INPUT F1(X(1),...,X(N))
EXP X(2)/X(1)-1	INPUT INITIAL X(1)
1	INPUT F2(X(1),...,X(N))
X(1)+X(2)-2	INPUT INITIAL X(2)
1	(iteration 0 printed) CONTINUE?
{enter}	(iteration 1 printed) CONTINUE?
... keep iterating! ...	
{enter}	(iteration 7 printed) CONTINUE?
N	(program stops)

Step 4: Discussion and suggestions:

In this problem, the common zero of the two surfaces F1 and F2 is found. From F2 = $X_1 + X_2 - 2$, the zero must lie on the line

$$X_2 = 2 - X_1$$

From F1 = $EXP(X_2)/X_1 - 1$, the zero must also lie on the curve:

$$X_2 = LN(X_1)$$

The common zero is the intersection of these two curves, shown in figure 6-3.

Try different initial guesses, some from each quadrant of the X_1-X_2 coordinate system, shown in figure 6-3. Find out which values converge most rapidly to a solution, and which, if any, never converge.

Demonstration 6.3 NON-LINEAR EQUATION

Step 1: Enter program and subroutines: ROOTN MXINV MXMUT

Step 2 (PROBLEM): Find a solution of the following non-linear equation

COS(X) = X

with initial guess: X = 1

Step 3: Run program:

USER ENTERS:	COMPUTER RESPONDS:
GOTO 6000	INPUT NO. EQNS./VARS.
1	INPUT F1(X(1),...,X(N))
COS X(1)-X(1)	INPUT INITIAL X(1)
1	(iteration 0 printed) CONTINUE?
{enter}	(iteration 1 printed) CONTINUE?
... keep iterating! ...	
{enter}	(iteration 4 printed) CONTINUE?
N	(program stops)

Step 4: Discussion and suggestions:

The one-dimensional problem in step 2 is shown in figure 6-4. From the figure, there are many "local minima" in the function COS(X) - X. That is, the value of the function at a local minimum is less than the values of the function in the immediate vicinity. This may cause trouble for ROOTN, since the algorithm requires the X-intercept of a tangent line at X_I be closer to the root than X_I. However, as long as the initial guess is within some range close to the root, the requirement is met in this problem. Determine the range of values of initial guess which give a successful convergence of the program.

There are also problems where more than one zero exists. In this case, the initial guess is also critical to the outcome of the program. Try some problems with many zeros, such as F(X) = COS(X).

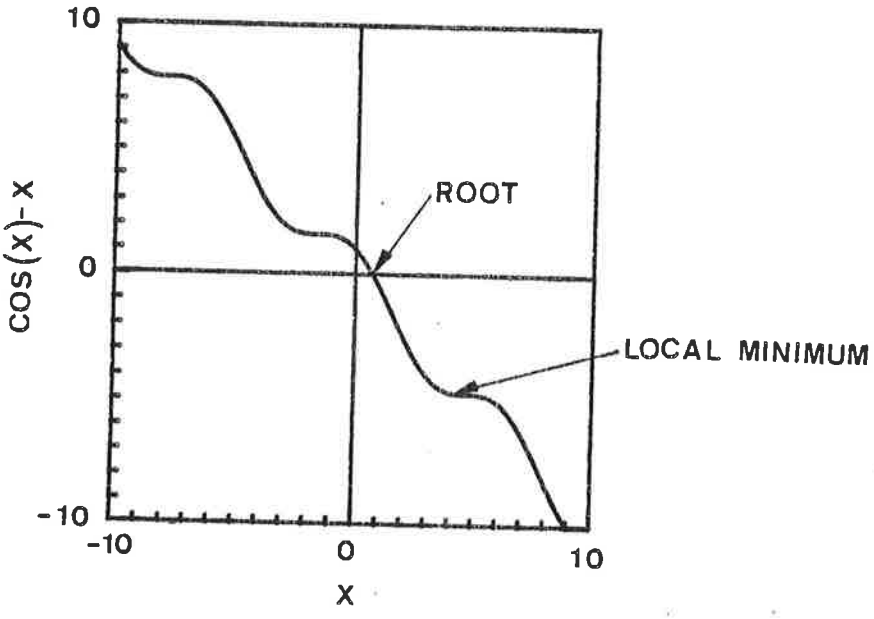


Figure 6-4 Root of COS(X)-X

```
6100 REM REGRP (LLS REGRESSION)
6102 DIM Z(N,N)
6104 DIM W(N)
6106 DIM G(N)
6108 FOR I=1 TO P
6110 FOR J=1 TO N
6112 LET G(J)=VAL F$(J)
6114 NEXT J
6116 FOR J=1 TO N
6118 FOR K=J+1 TO N
6120 LET Z(J,K)=Z(J,K)+G(J)*G(K)
6122 LET Z(K,J)=Z(J,K)
6124 NEXT K
6126 LET Z(J,J)=Z(J,J)+G(J)*G(J)
6128 LET W(J)=W(J)+G(J)*Y(I)
6130 NEXT J
6132 NEXT I
6134 LET X$="Z(I,J)"
6136 GO SUB MXINV
6138 LET M=N
6140 LET L=1
6142 LET X$="V(I,J)*W(J)"
6144 GO SUB MXMUT
6146 FOR J=1 TO N
6148 LET W(J)=V(J,1)
6150 NEXT J
6152 RETURN
```

Figure 6-5 Linear least-squares regression subroutine

REGRP

This subroutine performs a linear least-squares regression of a set of data points

$$\{(X_{1_1},\ldots,X_{E_1};Y_1),\ldots,(X_{1_P},\ldots,X_{E_P};Y_P)\}$$

to a function which is the linear sum of "basis functions", $\{F_I\}$

$$F(\vec{X}) = w_1 F_1(\vec{X}) + \ldots + w_N F_N(\vec{X})$$

where $\vec{X} = (X_1,\ldots,X_E)$

For example, a suitable basis function is $F_1(X_1,X_2) = 1 + X_1 + 2X_2$. The weights $w_1,\ldots,w_N$ are found, such that the error

$$\text{Error} = \sqrt{\sum_{I=1}^{P}[F(X_I) - Y_I]^2}$$

is minimized.

CALL:
- P = Number of data points
- E = Number of independent variables: $X_1,\ldots,X_E$
- N = Number of basis functions: $F_1,\ldots,F_N$
  = Number of weights: $w_1,\ldots,w_N$
- X(,) = Array of independent variable values of data points
- Y() = Array of dependent variable values of data points

REGRP: Changes G() I J K L M U U(,) V(,) W() X$ Z(,)

RETURN: W() = Array of weights for each basis function

Algorithm (linear least-squares regression):

Given P data points

$$\{(\vec{X}_1,Y_1),\ldots,(\vec{X}_P,Y_P)\} \quad ; \quad \vec{X}_I = (X_{1_I},\ldots,X_{E_I})$$

and N basis functions, $\{F_I(\vec{X})\}$; the parameters, $\{w_I\}$, for the regression function with the least error

$$F(\vec{X}) = w_1 F_1(\vec{X}) + \ldots + w_N F_N(\vec{X})$$

are given by

$$\begin{bmatrix} w_1 \\ \vdots \\ w_N \end{bmatrix} = \begin{bmatrix} \langle F_1,F_1\rangle & \ldots & \langle F_1,F_N\rangle \\ \vdots & & \vdots \\ \langle F_N,F_1\rangle & \ldots & \langle F_N,F_N\rangle \end{bmatrix}^{-1} \begin{bmatrix} \langle Y,F_1\rangle \\ \vdots \\ \langle Y,F_N\rangle \end{bmatrix}$$

where $\langle F_J,F_K\rangle = \sum_{I=1}^{P} F_J(X_I)*F_K(X_I)$; $\langle Y,F_J\rangle = \sum_{I=1}^{P} Y_I*F_J(X_I)$

Demonstration 6.4 GENERAL POLYNOMIAL REGRESSION

Step 1: Enter main program:

```
 530 REM DEMO6.4
 531 LET REGRP=6100
 532 LET MXINV=5000
 533 LET MXMUT=5100
 534 PRINT "INPUT NO. DATA PTS,"
 535 INPUT P
 536 PRINT "INPUT NO. BASIS FUNC
TIONS"
 537 INPUT N
 538 PRINT "INPUT NO. INDEPENDEN
T VARIABLES"
 539 INPUT E
 540 DIM F$(N,50)
 541 DIM X(E,P)
 542 DIM Y(P)
 543 FOR I=1 TO N
 544 PRINT "INPUT F";I;"(X(1,I),
...,X(N,I))"
 545 INPUT F$(I)
 546 NEXT I
 547 CLS
 548 FOR I=1 TO P
 549 FOR J=1 TO E
 550 PRINT "INPUT X";J;"(";I;")"
 551 INPUT X(J,I)
 552 NEXT J
 553 PRINT "_____"
 554 PRINT "INPUT Y(";I;")"
 555 INPUT Y(I)
 556 CLS
 557 NEXT I
 558 GO SUB REGRP
 559 PRINT "J: COEFF(J):";TAB 18
;"F(J):"
 560 FOR J=1 TO N
 561 PRINT AT J,0;J;TAB 3;W(J);T
AB 18;F$(J)
 562 NEXT J
 563 PRINT
 564 PRINT "I: Y(I):";TAB 18;"PO
LY(I):"
 565 FOR I=1 TO P
 566 LET M=0
 567 FOR J=1 TO N
 568 LET M=M+VAL F$(J)*W(J)
 569 NEXT J
 570 PRINT I;TAB 3;Y(I);TAB 18;M
 571 NEXT I
 572 STOP
```

Step 2: Enter subroutines: REGRP MXINV MXMUT

Step 3 (PROBLEM 1): Given the data in the table below, find the coefficients {W(I)} of the least-squares fit to the function

$$F(X_1,X_2) = W(1) + W(2)*EXP(-X_1) + W(3)*EXP(-X_2) + W(4)*EXP(-X_1*X_2)$$

I:	$X_1(I)$:	$X_2(I)$:	Y(I):
1	1	1.00	6.2
2	1	1.25	6.0
3	1	1.50	5.8
4	1	1.75	5.6
5	2	1.00	5.3
6	2	1.25	5.1
7	2	1.50	4.9
8	2	1.75	4.8

Step 4: Run program:

USER ENTERS:	COMPUTER RESPONDS:
RUN	INPUT NO. DATA PTS.
8	INPUT NO. BASIS FUNCTIONS
4	INPUT NO. INDEPENDENT VARS.
2	INPUT F1(X(1,I),...,X(N,I))
1	INPUT F2(X(1,I),...,X(N,I))
EXP -X(1,I)	INPUT F3(X(1,I),...,X(N,I))
EXP -X(2,I)	INPUT F4(X(1,I),...,X(N,I))
EXP (-X(1,I)*X(2,I))	INPUT X1(1)
1	INPUT X2(1)
1.00	INPUT Y(1)
6.2	INPUT X1(2)

... enter data from table in step 3 ...

1.75	INPUT Y(8)
4.8 {wait about 33s}	(results printed)

Step 5 (PROBLEM 2): Fit the data in the table below to a third-degree polynomial

$$F(X) = w_1 + w_2X + w_3X^2 + w_4X^3$$

I:	X(I):	Y(I):
1	.01	1.02
2	.05	1.11
3	.10	1.23
4	3.2	169
5	7.9	2180
6	1.1	12.2
7	2.0	49
8	5.2	655

Step 6: Run program:

USER ENTERS:	COMPUTER RESPONDS:
RUN	INPUT NO. DATA PTS.
8	INPUT NO. BASIS FUNCTIONS
4	INPUT NO. INDEPENDENT VARS.
1	INPUT F1(X(1,I),...,X(N,I))
1	INPUT F2(X(1,I),...,X(N,I))
X(1,I)	INPUT F3(X(1,I),...,X(N,I))
X(1,I)**2	INPUT F4(X(1,I),...,X(N,I))
X(1,I)**3	INPUT X1(1)
.01	INPUT Y(1)
1.02	INPUT X1(2)

... enter data from table in step 5 ...

5.2	INPUT Y(8)
655 (wait about 50s)	(results printed)

Step 7: Discussion:

The mean-squared error

$$\text{Error} = \sqrt{\sum_{I=1}^{P} \left(\sum_{J=1}^{N} w_J F_J(X_I) - Y_I \right)^2}$$

is minimized by REGRP. The choice of basis functions is, of course, critical to the regression problem. For example, a regression to one basis function, $F_1 = 1$, could be done in problems 1 and 2. A solution will be found, but the total error will be large even though it has been minimized. Therefore, a choice of basis functions should reflect the nature of the data, or an expected result. Since the data in problems 1 and 2 was fabricated, the choice of basis functions was simply those used to construct the data points.

The number of data points in both problems was rather small. For some regressions, this may be a problem, especially when the number of independent variables is large or the number of basis functions is large relative to the number of points.

Step 8: Suggestions:

Record some data. For example, take your pulse before, during, and after some stressful activity, (like a viewing of the 6:00 news). It might look like figure 6-6. Choose a number of basis functions which reflect the nature of the data, a displaced "hump", centered at time t_x:

$$F_1 = 1$$
$$F_2 = (t - t_x)$$
$$F_3 = (t - t_x)^2$$
$$\vdots$$

Use REGRP to find a regression curve, and plot it using PLOT2. If these basis functions don't work, choose some other set. You may want to regress part of the curve at a time. For example, the initial rise is much different from the final fall in figure 6-6. A better choice for the basis functions might be a set of exponentials.

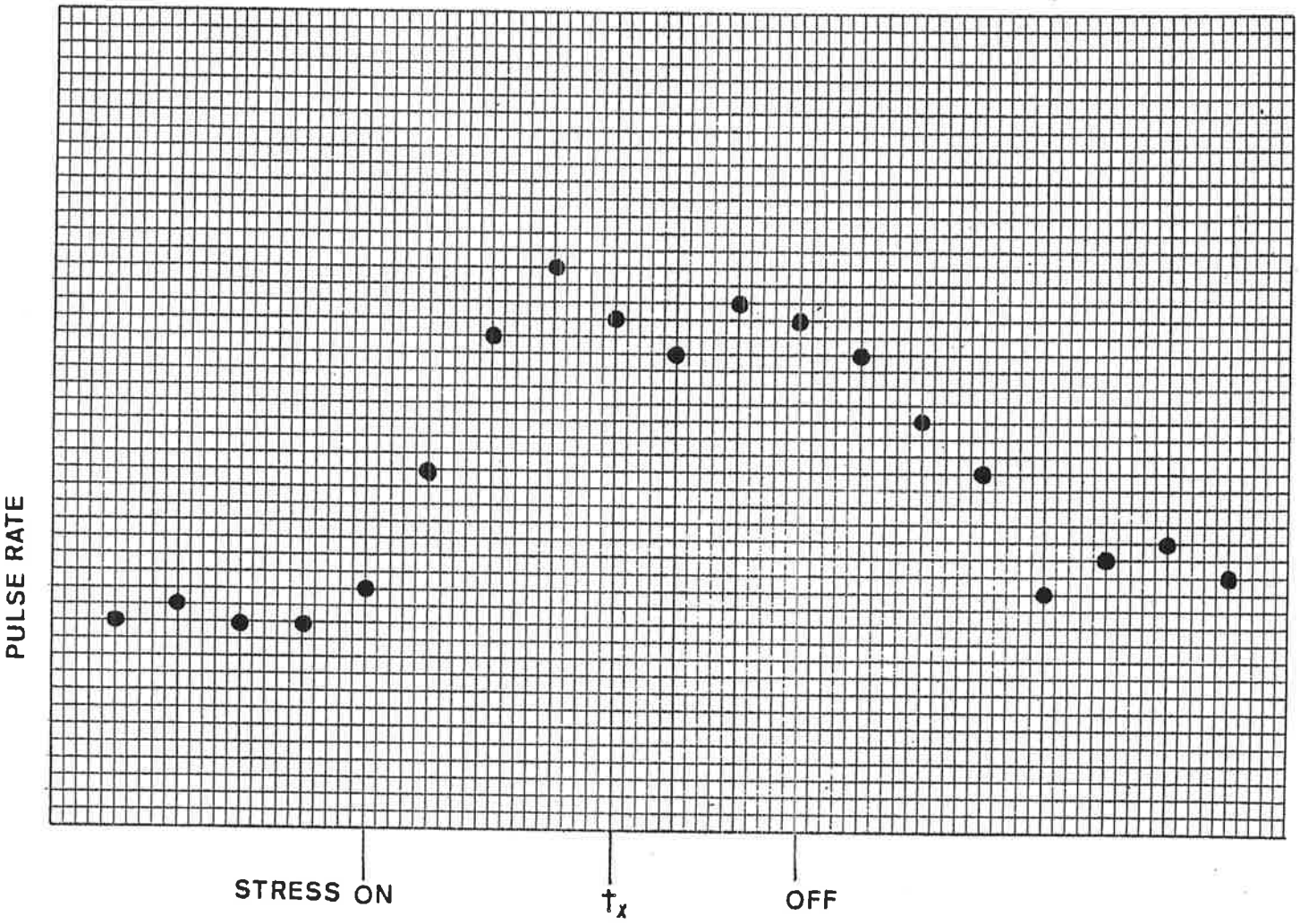


Figure 6-6 Pulse rate response to stress

Demonstration 6.5 LINEAR REGRESSION WITH PLOT

Step 1: Enter main program:

```
580 REM DEMO6.5
581 LET REGRP=6100
582 LET MXINV=5000
583 LET MXMUT=5100
584 LET POINT=2000
585 LET PLOTD=2400
586 LET VINAX=1000
587 LET SCALE=2300
588 GO SUB POINT
589 GO SUB PLOTD
590 LET P=N
591 LET N=2
592 DIM F$(N,4)
593 LET F$(1)="1"
594 LET F$(2)="X(I)"
595 GO SUB REGRP
596 PRINT "SLOPE = ";W(2)
597 PRINT "Y-INT = ";W(1)
598 FOR X=A TO B STEP (B-A)/100
599 LET Y=W(1)+W(2)*X
600 IF Y<C OR Y>D THEN GO TO 60
2
601 PLOT 240*(X-A)/(B-A),160*(Y
-C)/(D-C): REM *** FOR ZX81, USE
 *** PLOT 60*(X-A)/(B-A),40*(Y-C
)/(D-C)
602 NEXT X
603 STOP
```

Step 2: Enter subroutines: VINAX POINT.1 SCALE.2 PLOTD MXINV MXMUT REGRP

Step 3 (PROBLEM): Find the slope and y-intercept of the least-squares fitted line for the data in the table below:

I:	X(I):	Y(I):
1	2.0	0
2	3.0	0
3	3.2	0
4	4.0	1.1
5	4.2	4.0
6	4.5	4.5
7	6.2	6.0
8	6.5	6.1
9	6.8	9.1
10	7.0	10

Step 4: Run program:

USER ENTERS:	COMPUTER RESPONDS:
RUN	INPUT NO. PTS.
10	X(1),Y(1)=?
2.0	
0	X(2),Y(2)=?

... enter data from table ...

9.1	X(10),Y(10)=?
7.0	
10 (wait about 15s)	(results printed, plotted)

Step 5: Discussion and suggestions:

The linear-regression problem, as opposed to the problems of demonstration 6.4, have the two basis functions

$$F_1 = 1$$
$$F_2 = X$$

The regression curve is a straight line:

$$F(X) = w_1 + w_2 * X$$

and w_1 is the y-intercept and w_2 is the slope.

Exponential and power series regressions may be done simply by entering the appropriate basis functions into lines 593 and 594. Use different basis functions to regress the data in step 3. Compute the error for each set of basis functions, (see spec-sheet for REGRP), and see which choice best fits the data.

Find some real-world data for a linear regression, or an exponential regression. The regression line or curve may be used to make a prediction, given the past set of data points.

```
6200 REM REGRN (NLLS REGRESSION)
6202 DIM D(N)
6204 FOR J=1 TO N
6206 PRINT "INPUT INITIAL PAR(";
J;")"
6208 INPUT W(J)
6210 NEXT J
6212 LET Q=-1
6214 LET Q=Q+1
6216 CLS
6218 PRINT "ITER. ";Q
6220 PRINT "J:  PAR(J):"
6222 FOR J=1 TO N
6224 PRINT J;TAB 4;W(J)
6226 LET D(J)=(W(J)+(ABS W(J)<.0
01))*.001
6228 NEXT J
6230 PRINT "----------"
6232 PRINT "CONTINUE?"
6234 INPUT X$
6236 IF X$="N" THEN RETURN
6238 DIM Z(N,N)
6240 DIM B(N)
6242 DIM G(N)
6244 FOR I=1 TO P
6246 LET F=VAL F$
6248 FOR J=1 TO N
6250 LET M=W(J)
6252 LET W(J)=M+D(J)
6254 LET G(J)=(VAL F$-F)/D(J)
6256 LET W(J)=M
6258 NEXT J
6260 FOR J=1 TO N
6262 FOR K=J+1 TO N
6264 LET Z(J,K)=Z(J,K)+G(J)*G(K)
6266 LET Z(K,J)=Z(J,K)
6268 NEXT K
6270 LET Z(J,J)=Z(J,J)+G(J)*G(J)
6272 LET B(J)=B(J)+G(J)*(Y(I)-F)
6274 NEXT J
6276 NEXT I
6278 LET X$="Z(I,J)"
6280 GO SUB MXINV
6282 LET M=N
6284 LET L=1
6286 LET X$="V(I,J)*B(J)"
6288 GO SUB MXMUT
6290 FOR J=1 TO N
6292 LET W(J)=W(J)+U(J,1)
6294 NEXT J
6296 GO TO 6214
```

Figure 6-7 Non-linear least-squares regression subroutine

REGRN

This subroutine performs a non-linear least-squares regression of a set of data points

$$\{(X_{1_1},\ldots,X_{E_1};Y_1),\ldots,(X_{1_P},\ldots,X_{E_P};Y_P)\}$$

to a function containing N parameters $w_1,\ldots,w_N$ For example, $F(X_1,X_2) = w_1 X_1 + w_2 X_2 + w_3 X_1 X_2$ contains 3 parameters.

The weights $w_1,\ldots,w_N$ are found, such that the error

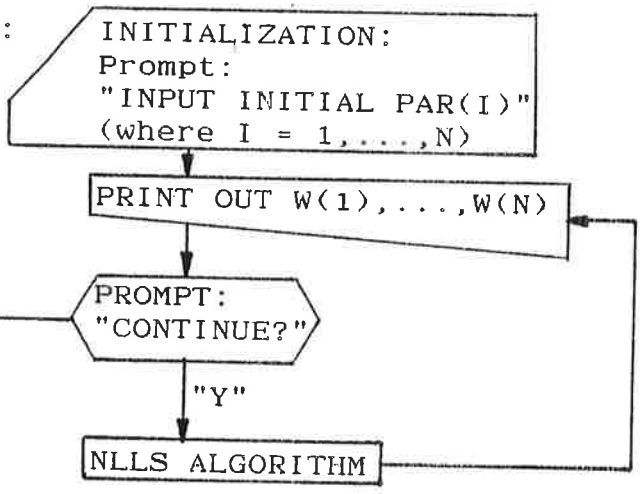
$$\text{Error} = \sqrt{\sum_{I=1}^{P}[F(X_I) - Y_I]^2}$$

is minimized.

An initial guess for the parameter values is entered, and the subroutine iterates to a solution. At each iteration, the current parameter values are printed. Convergence is indicated by non-changing parameter values.

CALL:
- P = Number of data points
- E = Number of independent variables: $X_1,\ldots,X_E$
- N = Number of parameters: $w_1,\ldots,w_N$
- F$ = Regression function: $F(X_1,\ldots,X_E;w_1,\ldots,w_N)$
- X(,) = Array of independent variable values of data points
- Y() = Array of dependent variable values of data points
- W() = Array to be returned from REGRP with parameter values. Must be dimensioned
  DIM W(N)

REGRN:



RETURN: W() = Regression coefficients

Algorithm (non-linear least-squares, thanks to Anant K. Agarwal):

Given:

1. P data points

$$\{(\vec{X}_1, Y_1), \ldots, (\vec{X}_P, Y_P)\} \quad ; \quad \vec{X}_I = (X_{1_I}, \ldots, X_{E_I})$$

2. An approximation function

$$F(\vec{X}; \vec{W}) \quad ; \quad \vec{W} = (w_1, \ldots, w_N)$$

3. A current (L^{th}) iteration estimate for the parameter vector: $\vec{W}_L$

The new estimate, $\vec{W}_{L+1}$, is

$$\vec{W}_{L+1} = \vec{W}_L + \begin{bmatrix} \langle \frac{\partial F}{\partial w_1}, \frac{\partial F}{\partial w_1} \rangle & \ldots & \langle \frac{\partial F}{\partial w_1}, \frac{\partial F}{\partial w_N} \rangle \\ \vdots & & \vdots \\ \langle \frac{\partial F}{\partial w_N}, \frac{\partial F}{\partial w_1} \rangle & \ldots & \langle \frac{\partial F}{\partial w_N}, \frac{\partial F}{\partial w_N} \rangle \end{bmatrix}^{-1} \begin{bmatrix} \langle Y-F, \frac{\partial F}{\partial w_1} \rangle \\ \vdots \\ \langle Y-F, \frac{\partial F}{\partial w_N} \rangle \end{bmatrix}$$

where $\langle \frac{\partial F}{\partial w_J}, \frac{\partial F}{\partial w_K} \rangle = \sum_{I=1}^{P} \frac{\partial F(X_I)}{\partial w_J} * \frac{\partial F(X_I)}{\partial w_K}$

$$\langle Y-F, \frac{\partial F}{\partial w_J} \rangle = \sum_{I=1}^{P} [Y_I - F(X_I)] * \frac{\partial F(X_I)}{\partial w_J}$$

Demonstration 6.6 GENERAL NON-LINEAR LEAST-SQUARES REGRESSION

Step 1: Enter main program:

```
 610 REM DEMO6.6
 611 LET REGRN=6200
 612 LET MXINV=5000
 613 LET MXMUT=5100
 614 PRINT "INPUT NO. DATA PTS."
 615 INPUT P
 616 PRINT "INPUT NO. PARAMETERS
"
 617 INPUT N
 618 PRINT "INPUT NO. INDEPENDEN
T VARIABLES"
 619 INPUT E
 620 DIM W(N)
 621 DIM X(E,P)
 622 DIM Y(P)
 623 PRINT "INPUT F(X(1,I),...,X
(E,I);W(1),...,W(N))"
 624 INPUT F$
 625 FOR I=1 TO P
 626 CLS
 627 FOR J=1 TO E
 628 PRINT "INPUT X";J;"(";I;")"
 629 INPUT X(J,I)
 630 NEXT J
 631 PRINT "_____"
 632 PRINT "INPUT Y(";I;")"
 633 INPUT Y(I)
 634 NEXT I
 635 GO SUB REGRN
 636 STOP
```

Step 2: Enter subroutines: MXINV MXMUT REGRN

Step 3 (PROBLEM 1): Fit the data in the table below to the function

$$F(X) = (1 + w_1 X)^{w_2}$$

I:	X(I):	Y(I):
1	.20	1.02
2	.40	1.05
3	.60	1.07
4	.80	1.09
5	1.0	1.11
6	1.2	1.12
7	1.4	1.14
8	1.6	1.16

assume initially $w_1 = w_2 = 1$

Step 4: Run program:

USER ENTERS:	COMPUTER RESPONDS:
RUN	INPUT NO. DATA PTS.
8	INPUT NO. PARAMETERS
2	INPUT NO. INDEPENDENT VARIABLES
1	INPUT F(X(1,I),...,X(E,I);W(1),...,W(N))
(1+W(1)*X(1,I))**W(2)	INPUT X1(1)
.2	INPUT Y(1)
1.02	INPUT X1(2)
... enter data from table in step 3 ...	
1.6	INPUT Y(8)
1.16	INPUT INITIAL PAR(1)
1	INPUT INITIAL PAR(2)
1	(iteration 0 printed) CONTINUE?
{enter}	(iteration 1 printed) CONTINUE?
... keep iterating! ...	
{enter}	(iteration 8 printed) CONTINUE?
N	(program stops)

Step 5 (PROBLEM 2): Fit the data in the table below to the function

$$F(X_1,X_2) = \frac{w_1 (X_1 - w_2) X_2}{1 + w_3 X_1 + w_4 X_2}$$

I:	X_1:	X_2:	Y:
1	2.5	2.5	3.3E-6
2	2.5	5.0	6.6E-6
3	2.5	7.5	9.9E-6
4	2.5	10.0	1.3E-5
5	2.5	12.5	1.6E-5
6	5.0	2.5	8.0E-6
7	5.0	5.0	1.6E-5
8	5.0	7.5	2.4E-5
9	5.0	10.0	3.1E-5
10	5.0	12.5	3.9E-5

Assume initially: w_1=.01,w_2=.1,w_3=.1,w_4=.01

Step 6: Run program:

USER ENTERS:	COMPUTER RESPONDS:
RUN	INPUT NO. DATA PTS.
10	INPUT NO. PARAMETERS
4	INPUT NO. INDEPENDENT VARIABLES
2	INPUT F(X(1,I),...,X(E,I);W(1),...,W(N))
W(1)*(X(1,I)-W(2))*X(2,I)/(1+W(3)*X(1,I)+W(4)*X(2,I))	
	INPUT X1(1)
2.5	INPUT X2(1)
2.5	INPUT Y(1)
3.3E-6	INPUT X1(2)

... enter data from table in step 5 ...

12.5	INPUT Y(10)
3.9E-5	INPUT INITIAL PAR(1)
.01	INPUT INITIAL PAR(2)
.1	INPUT INITIAL PAR(3)
.1	INPUT INITIAL PAR(4)
.01	(iteration 0 printed) CONTINUE?
{ENTER}	(iteration 1 printed) CONTINUE?

... keep iterating! ...

{enter}	(iteration 7 printed) CONTINUE?
N	(program stops)

Step 7: Discussion and suggestions:

The non-linear least squares problem usually requires many iterations before a solution is found. There are many cases when convergence does not occur or when the parameters get set to unrealistic values. That is, it is trickier to use REGRN than REGRP. On the other hand, if a theoretical formula is to be applied to a set of data points and approximate values of the fitting parameters are already known, then REGRN may give good "optimized" values for the parameters. In this problem, as in the root-finding problem solved by ROOTN, the choice of initial values for the parameters is very critical: the results from one initial guess may differ radically from the results obtained starting with another initial guess.

The demonstration might be improved by limiting the values of the calculated parameters. For example, if the "realistic" values that might be taken on by a parameter w_I are between 0 and 2 then the subroutine would limit the value of w_I to between 0 and 2 at each iteration.

Use some different data, plot it using PLOT2, (Chapter 2), and try to fit it with various functions using REGRN.

Demonstration 6.7 EXPONENTIAL REGRESSION WITH PLOT

Step 1 (PROBLEM): Given the data in the table below, find the parameters w_1 and w_2 of the least-squares fit to the function

$$F(X) = w_1 \, EXP(w_2 \, X)$$

I:	X(I):	Y(I):
1	.01	2
2	.22	4
3	.35	6
4	.53	12
5	.76	20
6	.81	23
7	.96	36
8	.98	38

Step 2: Enter main program:

```
 640 REM DEMO8.7
 641 LET REGRN=6200
 642 LET MXINV=5000
 643 LET MXMUT=5100
 644 LET POINT=2000
 645 LET PLOTD=2400
 646 LET VINAX=1000
 647 LET SCALE=2300
 648 GO SUB POINT
 649 LET P=N
 650 LET N=2
 651 DIM W(N)
 652 LET F$="W(1)*EXP (W(2)*X(I)
)"
 653 GO SUB REGRN
 654 CLS
 655 LET N=P
 656 GO SUB PLOTD
 657 FOR X=A TO B STEP (B-A)/100
 658 LET Y=W(1)*EXP (W(2)*X)
 659 IF Y<C OR Y>D THEN GO TO 66
1
 660 PLOT 240*(X-A)/(B-A),160*(Y
-C)/(D-C): REM *** FOR ZX81, USE
 *** PLOT 60*(X-A)/(B-A),40*(Y-C
)/(D-C)
 661 NEXT X
 662 STOP
```

Step 3: Enter subroutines: VINAX SCALE PLOTD POINT MXINV MXMUT REGRN

Step 4: Run program:

USER ENTERS:	COMPUTER RESPONDS:
RUN	INPUT NO. PTS.
8	X(1),Y(1)=?
.01	
2	X(2),Y(2)=?

... enter data from table in step 1 ...

36	X(8),Y(8)=?
.98	
38	INPUT INITIAL PAR(1)
1.5	INPUT INITIAL PAR(2)
2.5	(iteration 0 printed)
	CONTINUE?
{enter}	(iteration 1 printed)
	CONTINUE?

... keep iterating! ...

{enter}	(iteration 5 printed)
	CONTINUE?
N	(program stops)

Step 5: Discussion and suggestions:

This demonstration makes use of 7 subroutines, making for a short main program. Since the form of the fitting function is predetermined, fewer prompts are required than demonstration 6.6. After the parameters are found, a plot of the fitting function is done, along with the input points.

Try different initial guesses to determine if more than one set of "optimal" parameters exists.

Chapter 7

DIFFERENTIAL EQUATIONS

We all possess some ability to predict the future. When we do this, we synthesize our knowledge of the state of the universe at the present time with our knowledge of how it is likely to change, or how it has already been changing in the recent past. Thus, we solve a sort of boundary value problem when we perform such a prediction.

Differential equations are encountered in problems dealing with the change of a variable or variables in time or space, or with respect to some other set of parameters. The solution to such a problem is called a "trajectory": the predicted "path" in space, time, or some other parameter, taken by the variable. The boundary value problem consists of:

1. Knowledge of the present state of the "universe": The initial conditions. That is, the value of the variable at t=0 or x=0, or ...
2. Knowledge of the way the "universe" has been changing: The differential-equation(s) describing the mechanics of the problem.

The subroutine RKUTT solves a system of 1^{st} 4^{th} order differential equations, with initial conditions, using the 4^{th} order Runge-Kutta algorithm. RKUTT can deal with a vector of output variables, changing with respect to one input parameter. Differential equations involving higher ordered derivatives may be resolved into a system of 1^{st} order differential equations. An example is given in demonstration 7.1, problem 2.

```
7000 REM RKUTT (4TH ORDER RUNGE-
KUTTA)
7002 LET N=INT ((B-A)/H)+1
7004 DIM V(E,N)
7006 DIM K(E,4)
7008 FOR J=1 TO E
7010 LET V(J,1)=Y(J)
7012 NEXT J
7014 LET X=A
7016 FOR I=1 TO N-1
7018 LET K=1
7020 LET X$="V(J,I)"
7022 GO SUB 7048
7024 LET X=X+H/2
7026 LET X$="V(J,I)+K(J,K-1)/2"
7028 GO SUB 7048
7030 GO SUB 7048
7032 LET X=X+H/2
7034 LET X$="V(J,I)+K(J,3)"
7036 GO SUB 7048
7038 FOR J=1 TO E
7040 LET V(J,I+1)=V(J,I)+(K(J,1)
+K(J,4))/6+(K(J,2)+K(J,3))/3
7042 NEXT J
7044 NEXT I
7046 RETURN
7048 FOR J=1 TO E
7050 LET Y(J)=VAL X$
7052 NEXT J
7054 FOR J=1 TO E
7056 LET K(J,K)=H*VAL F$(J)
7058 NEXT J
7060 LET K=K+1
7062 RETURN
```

Figure 7-1 Differential equation solving subroutine

RKUTT

This subroutine numerically solves first-order vector differential equations of the form

$$\begin{bmatrix} \frac{dY_1(X)}{dX} \\ \vdots \\ \frac{dY_N(X)}{dX} \end{bmatrix} = \begin{bmatrix} F_1(X,Y_1(X),\ldots,Y_N(X)) \\ \vdots \\ F_N(X,Y_1(X),\ldots,Y_N(X)) \end{bmatrix}$$

over the interval: $X_L <= X <= X_U$. The values of the solution vector are specified at the lower end-point of the interval:

$$\begin{bmatrix} Y_1(X_L) \\ \vdots \\ Y_N(X_L) \end{bmatrix} = \begin{bmatrix} a_1 \\ \vdots \\ a_N \end{bmatrix}$$

The subroutine uses a fourth-order Runge-Kutta algorithm. Values for the solution vector are calculated at each step in the independent variable

$$X_I = X_L + (I-1)*H \;;\; I = 1,2,\ldots, \text{INT}((X_U - X_L)/H) + 1$$

where H is the user-supplied step-size for the integration.

CALL:
- E = Number of variables: $\{Y_I\}$
  = Number of equations: $\{Y_I' = F_I\}$
- A = Lower end-point: X_L
- B = Upper end-point: X_U
- H = Step-size
- F$() = Array of functions: $\{F_I\}$
- Y() = Values of Y_I at $X=X_L$

RKUTT:
- Changes I J K K(,) N V(,) X X$

RETURN:
- N = Number of steps
- V(,) = Solution at all steps:

  $V(J,I) = Y_J(X_L + (I-1)*H)$

Algorithm (4^{th}-order Runge-Kutta):

Given the solution point at a particular step

$$(X;Y_1,\ldots,Y_E)$$

the calculated solution point at the next step is

$$(X + \Delta X;Y_1 + \Delta Y_1,\ldots,Y_E + \Delta Y_E)$$

where

$$Y_J = 1/6 \left[K_1(J) + 2K_2(J) + 2K_3(J) + K_4(J)\right]$$

$$K_1(J) = \Delta X\ F_J(X;Y_1,\ldots,Y_E)$$

$$K_2(J) = \Delta X\ F_J(X+\tfrac{1}{2}\Delta X;Y_1+\tfrac{1}{2}K_1(1),\ldots,Y_E+\tfrac{1}{2}K_1(E))$$

$$K_3(J) = \Delta X\ F_J(X+\tfrac{1}{2}\Delta X;Y_1+\tfrac{1}{2}K_2(1),\ldots,Y_E+\tfrac{1}{2}K_2(E))$$

$$K_4(J) = \Delta X\ F_J(X+\Delta X;Y_1+K_3(1),\ldots,Y_E+K_3(E))$$

Demonstration 7.1 PLOT COMPUTED SOLUTIONS TO SEVERAL SYSTEMS OF DIFFERENTIAL EQUATIONS

Step 1: Enter main program:

```
 670 REM DEM07.1
 671 LET RKUTT=7000
 672 PRINT "INPUT NO. EQUATIONS"
 673 INPUT E
 674 PRINT "INPUT LOWER, UPPER L
IMITS"
 675 INPUT A
 676 INPUT B
 677 PRINT "INPUT STEP-SIZE"
 678 INPUT H
 679 DIM F$(E,50)
 680 DIM Y(E)
 681 FOR J=1 TO E
 682 PRINT "INPUT F";J;"(X,Y(1),
...,Y(N))"
 683 INPUT F$(J)
 684 PRINT "INPUT Y(";J;") AT X=
";A
 685 INPUT Y(J)
 686 CLS
 687 NEXT J
 688 GO SUB RKUTT
 689 LET C=1E23
 690 LET D=-C
 691 FOR J=1 TO E
 692 FOR I=1 TO N
 693 LET K=V(J,I)
 694 IF K<C THEN LET C=K
 695 IF K>D THEN LET D=K
 696 NEXT I
 697 NEXT J
 698 FOR J=1 TO E
 699 FOR I=1 TO N
 700 PLOT 240*(I-1)/(N-1),160*(V
(J,I)-C)/(D-C): REM *** FOR ZX81
, USE *** PLOT 60*(I-1)/(N-1),40
*(V(J,I)-C)/(D-C)
 701 NEXT I
 702 PRINT J
 703 NEXT J
 704 PRINT AT 0,0;"FROM (";A;","
;C;") TO (";B;";";D;")"
 705 STOP
```

Step 2: Enter subroutines: RKUTT

Step 3 (PROBLEM 1): Plot the solution to the second-order system of differential equations:

$$\begin{bmatrix} \frac{dY_1(X)}{dX} \\ \frac{dY_2(X)}{dX} \end{bmatrix} = \begin{bmatrix} -4Y_1+Y_2^3 \\ -EXP(2X)*Y_1 \end{bmatrix} \begin{bmatrix} Y_1(0) \\ Y_2(0) \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

over the interval: $0<=X<=1$

Step 4: Run program:

USER ENTERS:	COMPUTER RESPONDS:
RUN	INPUT NO. EQNS.
2	INPUT LOWER, UPPER LIMITS
0	
1	INPUT STEP-SIZE
.02	INPUT F1(X,Y(1),...,Y(N))
-4*Y(1)+Y(2)**3	INPUT Y(1) AT X=0
1	INPUT F2(X,Y(1),...,Y(N))
-EXP (2*X)*Y(1)	INPUT Y(2) AT X=0
1 {wait about 1m50s}	(results plotted)

Step 5 (PROBLEM 2): Plot the computed solution to Bessel's equation

$$X^2 \frac{d^2Y(X)}{dX^2} + X \frac{dY(X)}{dX} + (X^2 - N^2)Y = 0$$

for N = 0, over the interval $0<X<=5$
To do this, let

$$Y_1 = Y$$

$$Y_2 = \frac{dY}{dX}$$

This gives the system of 1st-order differential equations

$$\frac{dY_1}{dX} = \frac{dY}{dX} = Y_2$$

$$\frac{dY_2}{dX} = \frac{d^2Y}{dX^2} = -\left[X \frac{dY}{dX} + (X^2 - N^2)Y\right]/X^2 = -Y_2/X - Y_1$$

Y_1 is the solution to the original differential equation. Using the initial conditions $Y_1(0) = 1$; $Y_2(0) = 0$, the solution will be the zeroth-order Bessel's function of the first kind, J_0, (see chapter 3). Also, since $J_0' = -J_1$, Y_2 will be the inverted 1st-order Bessel's function of the first kind.

Step 6: Run program:

USER ENTERS:	COMPUTER RESPONDS:
RUN	INPUT NO. EQNS.
2	INPUT LOWER, UPPER LIMITS
1E-7	
5	INPUT STEP-SIZE
.05	INPUT F1(X,Y(1),...,Y(N))
Y(2)	INPUT Y(1) AT X=1E-7
1	INPUT F2(X,Y(1),...,Y(N))
-Y(2)/X-Y(1)	INPUT Y(2) AT X=1E-7
0 {wait about 2m20s}	(results plotted)

Step 7: Discussion and suggestions:

Problem 1 and problem 2 were both second-order problems. Problem 2 showed how a differential equation involving both first and second order derivatives may be resolved into two equations involving only the first derivative. Differential equations involving higher order derivatives may also be resolved into a system of first-order differential equations in this way.

There are a number of algorithms which are used to numerically solve differential equations. The Runge-Kutta method integrates the functions $F_1,\ldots,F_N$ at each step in the independent variable, X, using a rule similar to Simpson's rule, (see chapter 4), starting from the initial values of the dependent variables $Y_1,\ldots,Y_N$.

For example, the 1^{st} order problem is:

$$\frac{dY(X)}{dX} = F(X,Y) \quad ; \quad Y(A) = Y_A$$

Integrating:

$$Y(X) = Y_A + \int_A^{A+\Delta X} F(\hat{X},\hat{Y})d\hat{X} + \int_{A+\Delta X}^{A+2\Delta X} F(\hat{X},\hat{Y})d\hat{X} + \ldots + \int_{A+I\Delta X}^{X} F(\hat{X},\hat{Y})d\hat{X}$$

where $\hat{Y}$ is the value of Y in the small interval. Each integral is calculated using a rule. The integrations are continued in this way until X reaches its upper limit, B.

Use RKUTT to solve some other differential equations. Try some higher order systems of differential equations, and some differential equations involving higher orders of derivatives.

The general resolution of an N^{th}-order problem into a system of 1^{st}-order differential equations is:

1. Original equation:

$$\sum_{I=0}^{N} G_I(X)\,\frac{d^I Y(X)}{dX^I} + F(X) = 0$$

2. The N first-order equations are:

$$\frac{dY_1}{dX} = Y_2$$

$$\frac{dY_2}{dX} = Y_3$$

$$\vdots$$

$$\frac{dY_N}{dX} = -\left[\frac{F(X)}{G_N(X)} + \sum_{I=1}^{N}\frac{G_{I-1}(X)}{G_N(X)}\,Y_N\right]$$

The initial conditions are specified by the N^{th}-order derivatives of the function Y at A: $Y_N(A) = Y^{(N)}(A)$

Demonstration 7.2 PLOT COMPUTED SOLUTIONS TO SEVERAL FIRST-ORDER DIFFERENTIAL EQUATIONS

Step 1: Enter main program:

```
 710 REM DEMO7.2
 711 LET RKUTT=7000
 712 LET VINAX=1000
 713 LET SCALE=2200
 714 LET PLOTD=2400
 715 LET E=1
 716 DIM F$(E,50)
 717 DIM Y(E)
 718 PRINT "INPUT INTEGRAND=F(X,
Y(1))"
 719 INPUT F$(1)
 720 PRINT "INPUT LOWER, UPPER L
IMITS"
 721 INPUT A
 722 INPUT B
 723 PRINT "INPUT Y(1) AT X=";A
 724 INPUT Y(1)
 725 LET H=(B-A)/50
 726 GO SUB RKUTT
 727 DIM X(N)
 728 DIM Y(N)
 729 FOR I=1 TO N
 730 LET X(I)=A+H*(I-1)
 731 LET Y(I)=V(1,I)
 732 NEXT I
 733 GO SUB PLOTD
 734 STOP
```

Step 2: Enter subroutines: RKUTT VINAX SCALE.1 PLOTD

Step 3 (PROBLEM 1): Plot the computed solution to

$$\frac{dY}{dX} = -XY \quad ; \quad Y(-5) = EXP(-12.5) \quad ; \quad -5<=X<=5$$

Step 4: Run program:

USER ENTERS:	COMPUTER RESPONDS:
RUN	INPUT INTEGRAND=F(X,Y(1))
-X*Y(1)	INPUT LOWER, UPPER LIMITS
-5	
5	INPUT Y(1) AT X=-5
EXP -12.5 {wait about 50s}	(results plotted)

Step 5 (PROBLEM 2): Plot the computed solution to

$$\frac{dY}{dX} = -XY^2 \quad ; \quad Y(1) = 2 \quad ; \quad 1<=X<=2$$

Step 6: Run program:

USER ENTERS:	COMPUTER RESPONDS:
RUN	INPUT INTEGRAND=F(X,Y(1))
-X*Y(1)**2	INPUT LOWER, UPPER LIMITS
1	
2	INPUT Y(1) AT X=1
2 (wait about 1m15s)	(results plotted)

Step 7 (PROBLEM 3): Plot the computed solution to

$$\frac{dY}{dX} = -X/Y \quad ; \quad Y(0) = -1 \quad ; \quad 0<=X<=1$$

Step 8: Run program:

USER ENTERS:	COMPUTER RESPONDS:
RUN	INPUT INTEGRAND=F(X,Y(1))
-X/Y(1)	INPUT LOWER, UPPER LIMITS
0	
1	INPUT Y(1) AT X=0
-1 (wait about 50s)	(results plotted)

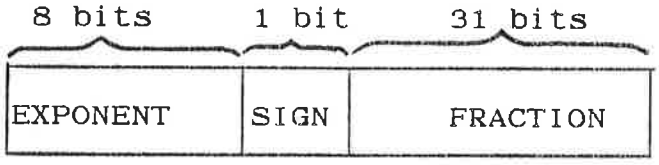
Step 9: Discussion and suggestions:

The last demonstration main program could also have been used to plot out the computed solutions to these problems. However, less prompts are required for this demonstration, since the order of the system of equations is pre-determined.

The problems in this demonstration may be solved by simple analytical methods, such as separation of variables. Compare the numerical results with the analytical results. Use different step-sizes, and observe the effect of varying the step-size on the accuracy of the computed results.

Chapter 8 NUMBER REPRESENTATIONS

There are many ways to symbolically represent a number. The ZX81 and TS2068 use this internal representation for "floating-point" numbers:



where a "bit" is either 0 or a 1. To get back to scientific notation, assuming EXPONENT, SIGN, and FRACTION are all integers in base 2 notation:

$$\text{number} = (-1)\uparrow(\text{SIGN}_{10})*(\text{FRACTION}_{10}*2^{-32}+2^{-1})*2\uparrow(\text{EXPONENT}_{10}-128)$$

where the subscripts "10" indicate base 10 notation corresponding to the base 2 integers.

This chapter presents a few other ways to represent numbers, including ratios of integers, 9's complement notation, and different base notations.

RATIO is a program that finds two integers whose ratio is close to or equal to the value of a given number. The real number system is composed of both rational and irrational numbers. The rational numbers are those whose values are given exactly by a ratio of integers. Since all numbers are represented by the ZX81 and TS2068 with only up to nine digits (base 10), all of the numbers which are represented on the computer are essentially rational: for example,

.123456789 = 123,456,789/1,000,000,000

RATIO, however, looks for the smallest integers whose ratio is within 10^{-6} of the given number.

The subroutine OPERA extends the accuracy of the computer for multiplications, additions, and subtractions. Operations with very large decimal-place accuracy take much longer than the usual 9-digit accuracy operations, (since the program is in BASIC, rather than in assembly language). The subroutine uses a similar representation of each number to the ZX81/TS2068 internal representation. However, base 10 notation is used rather than base 2. OPERA is used in demonstration 8.2, a reverse-polish notation calculator, with user-defined precision.

The program BASER changes the representation of a number in one base to another base. Both integers and fractions are handled by BASER, and fractions are truncated to the same accuracy as the original representation.

Finally, ZXCAL turns the ZX81/TS2068 into a scientific notation calculator, with successive results scrolled onto the screen.

```
8000 REM PROGRAM RATIO (RATIO-FI
NDER)
8002 PRINT "INPUT NUMBER ";
8004 INPUT E
8006 PRINT E
8008 LET A=1
8010 LET B=A
8012 LET C=A/B
8014 IF ABS (C-E)>1E-6 THEN GO T
O 8024
8016 PRINT "----------"
8018 PRINT A;"/";B;" = ";C
8020 PRINT "----------"
8022 STOP
8024 IF C<E THEN LET A=A+1
8026 IF C>E THEN LET B=B+1
8028 GO TO 8012
```

Figure 8-1 Ratio-finding program

RATIO

This program finds a ratio of two positive integers which approximates a user-supplied positive real number:

$$\left| \frac{A}{B} - E \right| <= 10^{-6}$$

where A and B are integers, and E is the user-supplied real number.

RUN:

RATIO: Prompt:
"INPUT NUMBER"

STOP: E = Input number
A = Numerator
B = Denominator

E A B and A/B printed

Demonstration 8.1 FIND RATIOS FOR SEVERAL IRRATIONAL NUMBERS, ACCURATE TO 6 DECIMAL PLACES

Step 1: Enter program: RATIO

Step 2 (PROBLEM 1): Find a ratio for ¶.

Step 3: Run program:

USER ENTERS:	COMPUTER RESPONDS:
GOTO 8000	INPUT NUMBER
PI (wait about 45s)	(results printed)

Step 4 (PROBLEM 2): Find a ratio for e.

Step 5: Run program:

USER ENTERS:	COMPUTER RESPONDS:
GOTO 8000	INPUT NUMBER
EXP 1 (wait about 5m45s)	(results printed)

Step 6 (PROBLEM 3): Find a ratio for $\sqrt{3/2}$.

Step 7: Run program:

USER ENTERS:	COMPUTER RESPONDS:
GOTO 8000	INPUT NUMBER
SQR 3/2 (wait about 3m15s)	(results printed)

Step 8: Discussion and suggestions:

It's interesting that the ratio of two fairly small integers gives a close approximation to the value of ¶. It takes about 7 times longer to find a ratio for e. There are, of course, many more irrational numbers which may be subjected to RATIO. In fact, the set of real numbers might be described as the set of irrational numbers with a few rational ones thrown in. However, as these examples show, given any irrational number, a rational number may be found which is arbitrarily close to it. This is fortunate, since computers cannot yet grasp the concept of the irrational number.

Try RATIO on an integer, a number with a repeating digit, such as

.0033333333... or .457457457...

and other numbers which may or may not be rational.

Find ratios with less accuracy than 10ppm by changing line 8014 of the program RATIO. Print out all results which are within a certain range of accuracy.

```
8100 REM OPERA (MULT. PREC. OPER
ATIONS)
8102 LET A=8*N
8104 LET B=LEN X$
8106 LET C=A+1
8108 LET D=0
8110 LET E=0
8112 LET F=1
8114 DIM X(A+2)
8116 LET X(A+1)=1
8118 IF X$(1 TO 1)="-" THEN LET
X(A+1)=-1
8120 FOR I=1 TO B
8122 LET A$=X$(I TO I)
8124 IF A$<>"." THEN GO TO 8130
8126 LET F=0
8128 GO TO 8152
8130 IF A$<>"E" THEN GO TO 8136
8132 LET E=E+VAL X$(I+1 TO B)
8134 GO TO 8154
8136 IF ABS (CODE A$-52)>5 THEN
GO TO 8152: REM *** FOR ZX81, US
E *** IF ABS (CODE A$-32)>5 THEN
 GO TO 8152
8138 LET G=VAL A$
8140 LET D=D+G
8142 IF D=0 THEN GO TO 8150
8144 LET C=C-1
8146 IF C<1 THEN GO TO 8152
8148 LET X(C)=G
8150 LET E=E-(NOT (D OR F))+(D>0
 AND F)
8152 NEXT I
8154 LET X(A+2)=E
8156 LET H=2
8158 IF Y$=">" THEN LET H=1
8160 FOR I=1 TO A+2
8162 LET Z(I,H)=X(I)
8164 NEXT I
8166 IF Y$=">" THEN GO TO 8500
8168 IF Y$="+" OR Y$="-" THEN GO
 TO 8300
8170 IF Y$="*" THEN GO TO 8200
8200 REM MULTX
8202 DIM Y(2*A)
8204 FOR J=1 TO A
8206 FOR I=1 TO A
8208 LET C=I+J-1
8210 LET Y(C)=Y(C)+Z(I,1)*Z(J,2)
8212 NEXT I
8214 NEXT J
8216 LET C=0
8218 FOR I=1 TO 2*A
8220 LET Y(I)=Y(I)+C
8222 LET C=INT (Y(I)/10)
8224 LET Y(I)=Y(I)-C*10
8226 NEXT I
8228 FOR I=1 TO 2*A
8230 LET J=2*A+1-I
8232 IF Y(J)<>0 THEN GO TO 8236
8234 NEXT I
8236 FOR I=1 TO A
8238 LET D=0
8240 IF I<J+1 THEN LET D=Y(J+1-I
)
8242 LET Z(A+1-I,1)=D
8244 NEXT I
8246 LET Z(A+1,1)=Z(A+1,1)*Z(A+1
,2)
8248 LET Z(A+2,1)=Z(A+2,1)+Z(A+2
,2)+J-2*A
8250 GO TO 8500
```

Figure 8-2 Multiple-precision operations subroutine

```
8300 REM ADSUB
8302 IF Y$="-" THEN LET Z(A+1,2)
=-Z(A+1,2)
8304 LET F=Z(A+2,2)-Z(A+2,1)
8306 LET H=(F>=0)+2*(F<0)
8308 LET F=ABS F
8310 GO SUB 8360
8312 FOR H=1 TO 2
8314 LET F=1
8316 GO SUB 8360
8318 LET D=Z(A+1,H)
8320 LET Z(A+1,H)=0
8322 IF D=-1 THEN GO SUB 8400
8324 NEXT H
8326 LET C=0
8328 FOR I=1 TO A+1
8330 LET Z(I,1)=Z(I,1)+Z(I,2)+C
8332 LET C=INT (Z(I,1)/10)
8334 LET Z(I,1)=Z(I,1)-C*10
8336 NEXT I
8338 LET H=1
8340 LET E=Z(A+1,1)
8342 IF E=9 THEN GO SUB 8400
8344 FOR I=1 TO A
8346 LET J=A+1-I
8348 IF Z(J,1)<>0 THEN GO TO 835
2
8350 NEXT I
8352 LET F=J-A
8354 GO SUB 8360
8356 LET Z(A+1,1)=(E=0)-(E=9)
8358 GO TO 8500
8360 REM SHIFT
8362 IF F=0 THEN RETURN
8364 DIM Y(3*A)
8366 FOR I=1 TO A
8368 LET Y(I+A)=Z(I,H)
8370 NEXT I
8372 FOR I=1 TO A
8374 LET D=0
8376 IF ABS F<=A THEN LET D=Y(I+
A+F)
8378 LET Z(I,H)=D
8380 NEXT I
8382 LET Z(A+2,H)=Z(A+2,H)+F
8384 RETURN
8400 REM COMPT
8402 FOR I=1 TO A+1
8404 LET Z(I,H)=9-Z(I,H)
8406 NEXT I
8408 LET C=1
8410 FOR I=1 TO A+1
8412 LET Z(I,H)=Z(I,H)+C
8414 LET C=INT (Z(I,H)/10)
8416 IF C=0 THEN RETURN
8418 LET Z(I,H)=Z(I,H)-C*10
8420 NEXT I
8422 RETURN
8500 REM DISPX
8502 IF Z(A+1,1)=-1 THEN PRINT "
-";
8504 PRINT ".";
8506 FOR I=1 TO A
8508 PRINT Z(A-I+1,1);
8510 NEXT I
8512 PRINT "E";Z(A+2,1)
8514 RETURN
```

OPERA

This subroutine performs multiple-precision multiplication, addition and subtraction of floating-point numbers. The precision is caller-defined. The subroutine is designed to be used in a Reverse-Polish Notation (RPN) calculator, as shown in demonstration 8.2.

CALL:
- N = Precision: number of decimal places = N×8
- Z(,) = Register array. Must be dimensioned DIM Z(8*N+2,2)
- X$ = Number in scientific notation
- Y$ = operation:
  - ">" for ENTER : Put number in accumulator
  - "+" for addition : Add number to accumulator
  - "-" for subtraction : Subtract number from accumulator
  - "*" for multiplication : Multiply number by accumulator

OPERA: Changes A A$ B C D E F G H I J X() Y() Z(,)

RETURN: Operation performed. Result is put in accumulator. Accumulator printed.

Summary of subroutine operation:

The register array has this format:

Register name:	Variable:	Format:
Accumulator	Z(*,1)	. F F F ... F F F | S | E
X	Z(*,2)	. F F F ... F F F | S | E

8N places

Where .FFF...FFF is the fractional part of the number, S is the sign of the number (±1) and E is the exponent of the number. The fractional part of the number is adjusted so the first number after the decimal point is non-zero.

Multiplication (MULTX):

Lines 8200 to 8214: Multiply all digits of accumulator with all digits of X and store result in Y().
Lines 8216 to 8226: Transfer portion of Y() that is greater than 9 to next digit, etc.
Lines 8228 to 8234: Look for first non-zero digit in Y().
Lines 8236 to 8244: Transfer Y() to accumulator such that first digit after decimal-point is non-zero.
Lines 8236 to 8248: Adjust sign and exponent.

Addition/Subtraction (ADSUB):

Line 8302: If operation is subtraction, then change sign of X and proceed with addition.
Lines 8304 to 8310: Depending upon difference in exponents of X and accumulator, determine the register to be shifted (H), and by how many places (F). Call shifting subroutine, (SHIFT).
Lines 8312-8324: Convert to ten's complement if sign of accumulator or X is negative.
Lines 8326 to 8336: Add X to accumulator.
Lines 8338 to 8342: Ten's complement accumulator if result is negative.
Lines 8344 to 8354: Left-shift accumulator to give a non-zero digit after the decimal point.
Line 8356: Adjust sign:
"0" = "+" and "9" = "-" to "+1" = "+" and "-1" = "-"

Demonstration 8.2 REVERSE-POLISH CALCULATOR

Step 1: Enter main program:

```
740 REM DEM08.2
741 LET OPERA=8100
742 PRINT "INPUT PRECISION"
743 INPUT N
744 DIM Z(8*N+2,2)
745 PRINT "INPUT NUMBER"
746 INPUT X$
747 PRINT "INPUT OPERATION"
748 INPUT Y$
749 CLS
750 GO SUB OPERA
751 GO TO 745
```

Step 2: Enter subroutines: OPERA

Step 3 (PROBLEM 1): Find the number of radians in one degree, to 24 decimal places.

radians/degree = ¶/180 = ¶ * $0.00555\overline{}$

Step 4: Run program:

USER ENTERS:	COMPUTER RESPONDS:
RUN	INPUT PRECISION
3	INPUT NUMBER
3.14159265358979238462643	INPUT OPERATION
>	(number is printed)
	INPUT NUMBER
.00555555555555555555555555	INPUT OPERATION
*	(result is printed)

Step 5 (PROBLEM 2): Using 15 place accuracy, compute the sum

$$S = 1 + X + X^2 + X^3 + X^4 + X^5 = X(X(X(X(X+1)+1)+1)+1)+1$$

for X = .005

Step 6: Run program:

USER ENTERS:	COMPUTER RESPONDS:
RUN	INPUT PRECISION
2	INPUT NUMBER
1	INPUT OPERATION
>	(accumulator is printed) INPUT NUMBER
.005	INPUT OPERATION
*	(accumulator is printed) INPUT NUMBER
1	INPUT OPERATION
+	(accumulator is printed) INPUT NUMBER
.005	INPUT OPERATION
*	(accumulator is printed) INPUT NUMBER
1	INPUT OPERATION
+	(accumulator is printed) INPUT NUMBER
.005	INPUT OPERATION
*	(accumulator is printed) INPUT NUMBER
1	INPUT OPERATION
+	(accumulator is printed) INPUT NUMBER
.005	INPUT OPERATION
*	(accumulator is printed) INPUT NUMBER
1	INPUT OPERATION
+	(accumulator is printed) INPUT NUMBER
.005	INPUT OPERATION
*	(accumulator is printed) INPUT NUMBER
1	INPUT OPERATION
+	(accumulator is printed)

Step 7: Discussion and suggestions

Problem 1 and problem 2 keep all of the interesting digits that are usually lost in computer calculations. Many multiple precision calculations are amazingly symmetrical; for example, those involving 9's.

Appendices B and C give some physical and mathematical constants and conversion factors with many decimal places. (The value for ¶ has been calculated to many more pages of decimal places than is shown in appendix B.) Many calculations can result in the amplification of round-off errors, and in these calculations, the larger the number of decimal places, the better.

Try writing a division subroutine for OPERA. Of course, multiple subtractions might be done, but this would take much time to perform.

```
8600 REM BASER (BASE-CONVERSION)
8602 PRINT "INPUT BASE (TO)"
8604 INPUT A
8606 PRINT "INPUT BASE (FROM)"
8608 INPUT B
8610 PRINT "INPUT NUMBER IN BASE
 ";B
8612 INPUT X$
8614 PRINT "....."
8616 PRINT X$;" BASE ";B
8618 LET M=LEN X$
8620 FOR K=1 TO M
8622 IF X$(K TO K)="." THEN GO T
O 8626
8624 NEXT K
8626 LET N=K-1
8628 IF N=0 THEN GO TO 8640
8630 DIM B(N)
8632 FOR I=1 TO K-1
8634 LET B(K-I)=CODE X$(I TO I)-
48: IF B(K-I)>9 THEN LET B(K-I)=
B(K-I)-7: REM *** FOR ZX81, USE
*** LET B(K-I)=CODE X$(I TO I)-2
8
8636 NEXT I
8638 GO SUB 8660
8640 PRINT ".";
8642 LET N=M-K
8644 IF N=0 THEN GO TO 8656
8646 DIM B(N)
8648 FOR I=K+1 TO M
8650 LET B(I-K)=CODE X$(I TO I)-
48: IF B(I-K)>9 THEN LET B(I-K)=
B(I-K)-7: REM *** FOR ZX81, USE
*** LET B(I-K)=CODE X$(I TO I)-2
8
8652 NEXT I
8654 GO SUB 8700
8656 PRINT " BASE ";A
8658 STOP
8660 REM INTBA
8662 DIM A(100)
8664 LET J=0
8666 LET J=J+1
8668 LET S=0
8670 LET R=0
8672 FOR I=N TO 1 STEP -1
8674 LET V=B(I)+R*B
8676 LET B(I)=INT (V/A)
8678 LET R=V-A*B(I)
8680 LET S=S+B(I)
8682 NEXT I
8684 LET A(J)=R
8686 IF S<>0 THEN GO TO 8666
8688 FOR I=J TO 1 STEP -1
8690 LET R=A(I)+48+7*(A(I)>9): P
RINT CHR$ R): REM *** FOR ZX81,
USE *** PRINT CHR$(A(I)+28);
8692 NEXT I
8694 RETURN
8700 REM FRABA
8702 DIM A(100)
8704 LET J=0
8706 LET J=J+1
8708 LET S=N*LN B/LN A
8710 LET R=0
8712 FOR I=N TO 1 STEP -1
8714 LET V=A*B(I)+R
8716 LET R=INT (V/B)
8718 LET B(I)=V-B*R
8720 NEXT I
8722 LET A(J)=R
8724 IF J<S THEN GO TO 8706
8726 FOR I=1 TO J
8728 LET R=A(I)+48+7*(A(I)>9): P
RINT CHR$ R): REM *** FOR ZX81,
USE *** PRINT CHR$ (A(I)+28);
8730 NEXT I
8732 RETURN
```

Figure 8-3 Base changing program

BASER

This program converts a number in one base notation to another base notation. The number may contain an integer part and a fraction part. Fractions are rounded off to the equivalent number of places in the new base notation to the old base notation.

RUN:

BASER: Prompts, in order of appearance:
1. "INPUT BASE(TO)"
2. "INPUT BASE(FROM)"
3. "INPUT NUMBER IN BASE B"
(where B is the old base)

STOP: Number printed old base and in new base

Example of algorithm:

1. Convert 12.34_{10} to base 2 notation.
2. Integer part:

Let $12 = a2^0 + b2^1 + c2^2 + d2^3 + \ldots$

Successive divisions of the integer part by 2 gives:

$6 = b2^0 + c2^1 + d2^2 + \ldots$; remainder a = 0

$3 = c2^0 + d2^1 + \ldots$; remainder b = 0

$1 = d2^0 + \ldots$; remainder c = 1

$0 = \ldots$; remainder d = 1

Hence, successive divisions by the new base gives remainders equal to the digits of the number in the new base, starting with the least-significant digit.

3. Fraction part:

Let $.34 = a2^{-1} + b2^{-2} + c2^{-3} + d2^{-4} + \ldots$

Successive multiplications of the fraction part by 2 gives:

$0.68 = a + b2^{-1} + c2^{-2} + d2^{-3} + \ldots$; integer part a = 0

$1.36 = b + c2^{-1} + d2^{-2} + \ldots$; integer part b = 1

$0.72 = c + d2^{-1} + \ldots$; integer part c = 0

$1.44 = d + \ldots$; integer part d = 1

4. From parts 2 and 3: $12.34_{10} = 1100.0101_2$

Demonstration 8.3 CONVERT SEVERAL NUMBERS FROM ONE BASE TO ANOTHER BASE

Step 1: Enter program: BASER

Step 2 (PROBLEM 1): Convert $F.A10C_{16}$ to base 10 notation.

Step 3: Run program:

USER ENTERS:	COMPUTER RESPONDS:
GOTO 8600	INPUT BASE(TO)
10	INPUT BASE(FROM)
16	INPUT NUMBER IN BASE 16
F.A10C	(results printed)

Step 4 (PROBLEM 2): Convert 10110.00111_2 to base 10 notation.

Step 5: Run program:

USER ENTERS:	COMPUTER RESPONDS:
GOTO 8600	INPUT BASE(TO)
10	INPUT BASE(FROM)
2	INPUT NUMBER IN BASE 2
10110.00111	(results printed)

Step 6 (PROBLEM 3): Find the base 16 representation of ¶.

Step 7: Run program:

USER ENTERS:	COMPUTER RESPONDS:
GOTO 8600	INPUT BASE(TO)
16	INPUT BASE(FROM)
10	INPUT NUMBER IN BASE 10
3.141592638979323846	(results printed)

Step 8 (PROBLEM 4): Find the base 2 representation of ¶.

Step 9: Run program:

USER ENTERS:	COMPUTER RESPONDS:
GOTO 8600	INPUT BASE(TO)
2	INPUT BASE(FROM)
10	INPUT NUMBER IN BASE 10
3.141592638979323846	(results printed)

Step 10: Discussion and suggestions:

Most of us are fluent only with base 10 representations of numbers. However, there are at least three other commonly used number representations: base 2 (Binary), base 8 (Octal), and base 16 (Hexadecimal). These bases are all powers of 2, and since all information in a digital computer exists, at least on some level, in a binary form, the binary, octal, and hexadecimal number systems are convenient representations to use. To change base notation within these powers-of-two representations is simple. For example, the change from base 2 to base 16:

```
base  2 number :  ... 1011 0110 . 0101 0101 ...
base 16 number :  ...  B    6   .  5    5   ...
```

the algorithm is to group four of the base 2 digits together, without overlapping the radix-point, and writing the base 16 digit corresponding to each group.

Note that an appropriate number of places is chosen in the new base notation by BASER. This corresponds to the same accuracy as the representation of the number in the old base.

It is highly recommended to view the representation of PI in base 32. Note that letters take the place of numbers after you reach base 9, (any base larger than 10). If you get higher than Z, then other symbols corresponding to character codes will be used.

```
8800 REM PROGRAM ZXCAL (ZX81 CAL
CULATOR)
8802 LET X$=" "
8804 REM *** FOR ZX81, USE *** S
CROLL
8806 INPUT Y$
8808 LET N=ABS (CODE Y$-32)
8810 IF N>12 THEN LET X$=Y$+X$
8812 IF N<12 THEN LET X$=X$+Y$
8814 IF N<6 THEN LET X$=Y$
8816 LET X$=STR$ VAL X$
8818 PRINT X$
8820 GO TO 8804
```

Figure 8-4 Scientific calculator

ZXCAL

This program makes the ZX81 or TS2068 emulate a scientific calculator with the functions:

+ - * / LN EXP SIN COS TAN ACS ASN ATN SQR

The "ENTER" key is equivalent to the "=" key on a scientific calculator. Successive results are scrolled onto the display.

Demonstration 8.4 SCIENTIFIC CALCULATOR

Step 1: Enter program: ZXCAL

Step 2 (PROBLEM): Use ZXCAL to calculate

EXP((355/113)*LN(3.2+6.8))

Step 3: Run program:

USER ENTERS:	COMPUTER RESPONDS:
GOTO 8800	(waits for input)
3.2 {enter}	3.2
+6.8 {enter}	10
LN {enter}	2.3025851
*355/113 {enter}	7.2337851
EXP {enter}	1385.4567

Step 4: Discussion and suggestions:

It is hoped that this book has shown that the computer is orders of magnitude more powerful than a hand-calculator. Hand-calculators are appreciated for their portability, but the computer can always take over when necessary, as ZXCAL shows.

ZXCAL may be included in other programs for a convenient calculator for stray calculations. If you have a printer, modify ZXCAL so that it prints out the successive results.

Appendix A

ONE-LINERS

Table A-1 lists some mathematical functions not found on the ZX81 and TS2068, but that may be implemented in one program line, as shown in the "Implement" column. The center column gives a range of values for the arguments, for those who like to stretch the limits of their machines.

Notes:

1. If the intent is: Y=min(X,Y), then this function may be more efficiently implemented by

   ```
   IF X<Y THEN LET Y=X
   ```

2. The MOD function gives the value of the argument in a particular MOD. For example, counting in MOD 5 goes like this:

   0 1 2 3 4 0 1 2 3 4 0 1 2 3 ...

Table A-1 One-liners

$\mu = 4\times10^{-39}$, $N = 1\times10^{38}$

INTENT:	CAREFUL!	IMPLEMENT:
$Z=\log_{10}(X)$	$\mu<X<N$	LET Z=LN X/LN 10
Z=cosh(X)	-88<X<88	LET Z=.5*(EXP X+EXP -X)
Z=sinh(X)	-88<X<88	LET Z=.5*(EXP X-EXP -X)
Z=tanh(X)	-N/10<X<44	LET Z=(EXP (2*X)-1)/(EXP (2*X)+1)
$Z=\cosh^{-1}(X)$	$1<=X<\sqrt{N}$	LET Z=LN (X+SQR (X*X-1))
$Z=\sinh^{-1}(X)$	$-10^{-7}<X<\sqrt{N}$	LET Z=LN (X+SQR (X*X+1))
$Z=\tanh^{-1}(X)$	-1<X<1	LET Z=.5*LN ((1+X)/(1-X))
① Z=min(X,Y)		LET Z=(Y<X)*Y+(Y>=X)*X
① Z=max(X,Y)		LET Z=(Y>X)*Y+(Y<=X)*X
② $Z=\mathrm{mod}_Y(X)$	X,Y integers	LET Z=X-INT (X/Y)*Y
Z=remainder(X/Y)		LET Z=X-INT (X/Y)*Y
Z=db conversion(X)	$\mu<X<N$	LET Z=20*LN X/LN 10
Z=deg to rad(X)		LET Z=X*PI/180
Z=rad to deg(X)		LET Z=X*180/PI
$Z=X^Y$	X may be <0	LET Z=SGN X*ABS X**Y
Z=truncation to Y plcs.(X)		LET Z=SGN X*INT (ABS X*10**Y)/10**Y

Appendix B PHYSICAL AND MATHEMATICAL CONSTANTS

Values of some of the more or less fundamental constants are shown in table B-1. The physical constants are from reference [22]. Uncertainties are given in parts per million (ppm) of the main number. For example, the uncertainty in the value of the speed of light is:

$$\pm(2.99792458\times10^{8})\times(.004\times10^{-6})\ \text{m/s} \doteq \pm1.2\text{m/s}$$

The mathematical constants are from references [1] and [3].

Table B-1 Constants

Physical (SI units)		
$N_A = 6.022045 \times 10^{23} \pm 5.1$ppm	$mole^{-1}$	Avogadro's Number
$c = 2.99792458 \times 10^{8} \pm .004$ppm	m/s	Speed of Light in Vacuum
$e = 1.6021892 \times 10^{-19} \pm 2.9$ppm	C	Proton Charge
$\hbar = 1.0545887 \times 10^{-34} \pm 5.4$ppm	J · s	Planck's Constant
$k_B = 1.380662 \times 10^{-23} \pm 32$ppm	J/K	Boltzmann Constant
$m_e = 9.109534 \times 10^{-31} \pm 5.1$ppm	kg	Electron rest mass
$G = 6.6720 \times 10^{-11} \pm 615$ppm	$N \cdot m^2/kg^2$	Gravitational Constant
Mathematical		
$\pi = 3.14159265358979323846264338327950288419716939937510582097494459230781640...$		
$e = 2.71828182845904523536028 7...$		
$\gamma = 0.57721566490153286060651 2...$		
$\ln 2 = 0.69314718055994530941723 21...$		
$\ln 10 = 2.30258509299404568401799 15...$		
$\log_{10} e = 0.43429448190325182765112 89...$		
$\sqrt{2} = 1.41421356237309504 88...$		

Appendix C

UNIT CONVERSIONS

The following series of conversion charts give conversion factors within and between metric and U.S. Customary systems of units. All conversion factors are exact except those noted, in which case the exact expression is found in the footnote. The charts show the units in their relative sizes on a logarithmic axis, with the smallest at bottom and the largest at top. For example, on the Length chart, there are 100 centimeters (cm) in a meter (m), so the label for the cm is spaced 2 units below the label for m, since $2 = \log_{10} 100$.

To obtain a conversion factor between any two units on a chart:

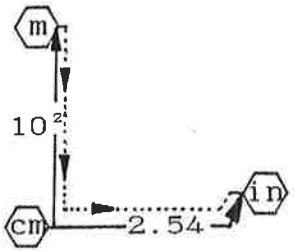
1. Locate the two units, UNIT1 and UNIT2,and find a path between them.

2. Let F=1
   Move along the path between UNIT1 and UNIT2:
   If moving in direction of arrow, then let F=F/NUMBER ON PATH
   If moving in opposition to arrow, then let F=F*NUMBER ON PATH
   At end of path, check: If UNIT2 is lower than UNIT1, then F should be greater than 1.

3. F has units of UNIT2/UNIT1:
   If converting X UNIT1's to Y UNIT2's, then let Y=X*F
   If converting Y UNIT2's to X UNIT1's, then let X=Y/F

4. Example: Convert 15.23 meters to inches.

   From the Length chart:



   Starting at meters (m) and moving toward inches along the dotted path:

   $F = 1*10^2/2.54$ in/m

   15.23m = 15.23*F in ≐ 599.61 in

Conversion Charts notes:

1. 6336/3937
2. $(3937/6336)^2 \times 6.4$
3. $.67200625 \times 1.6387064$
4. $(5659/8640) \times 1.9$
5. 175/144

Figure C-2 Length Conversion Factors

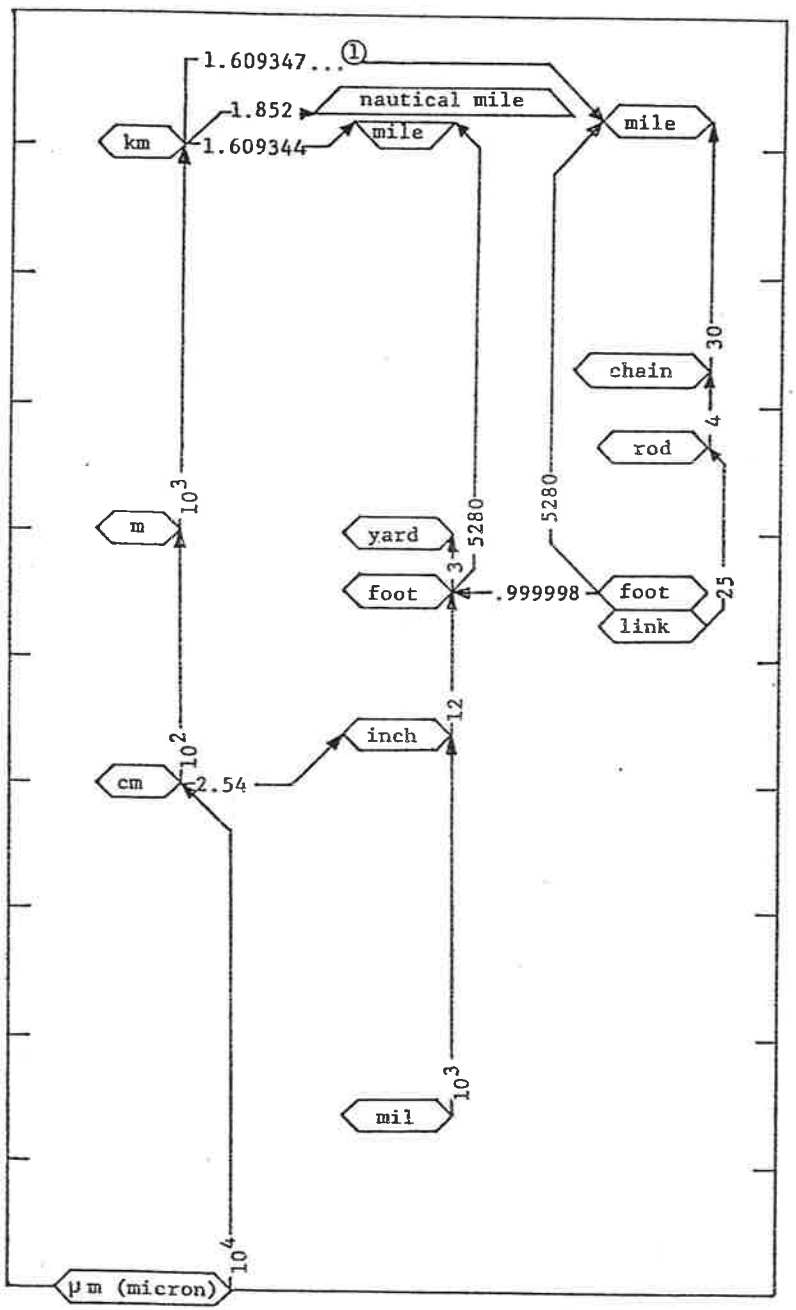


Figure C-1 Mass Conversion Factors

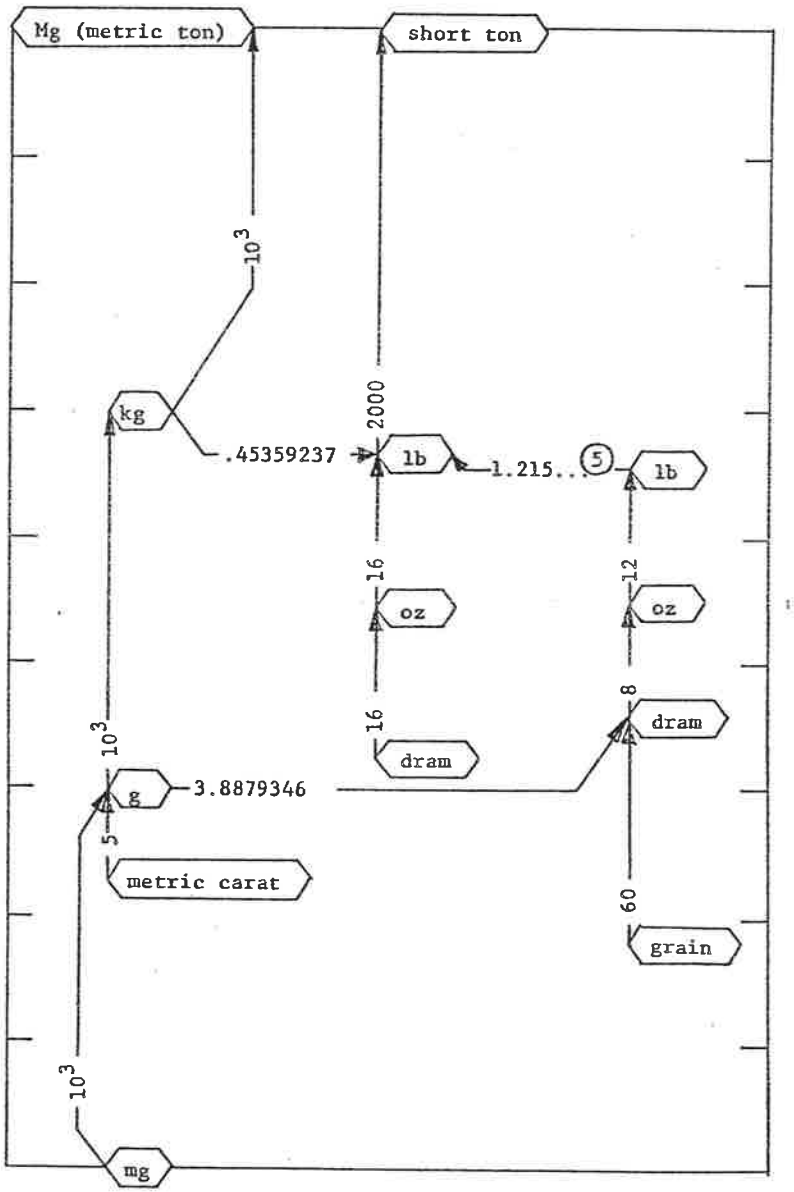


Figure C-4 Volume Conversion Factors

Metric US Customary
International Dry Liquid

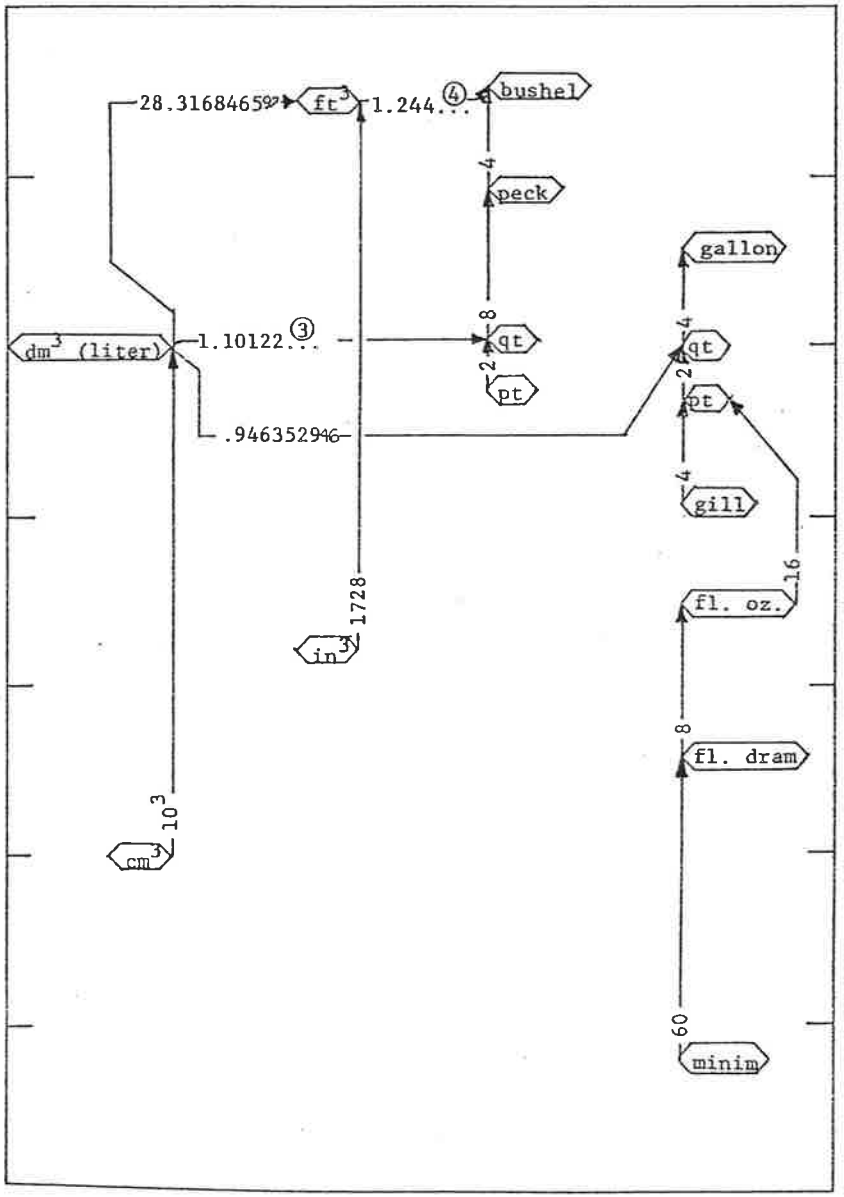
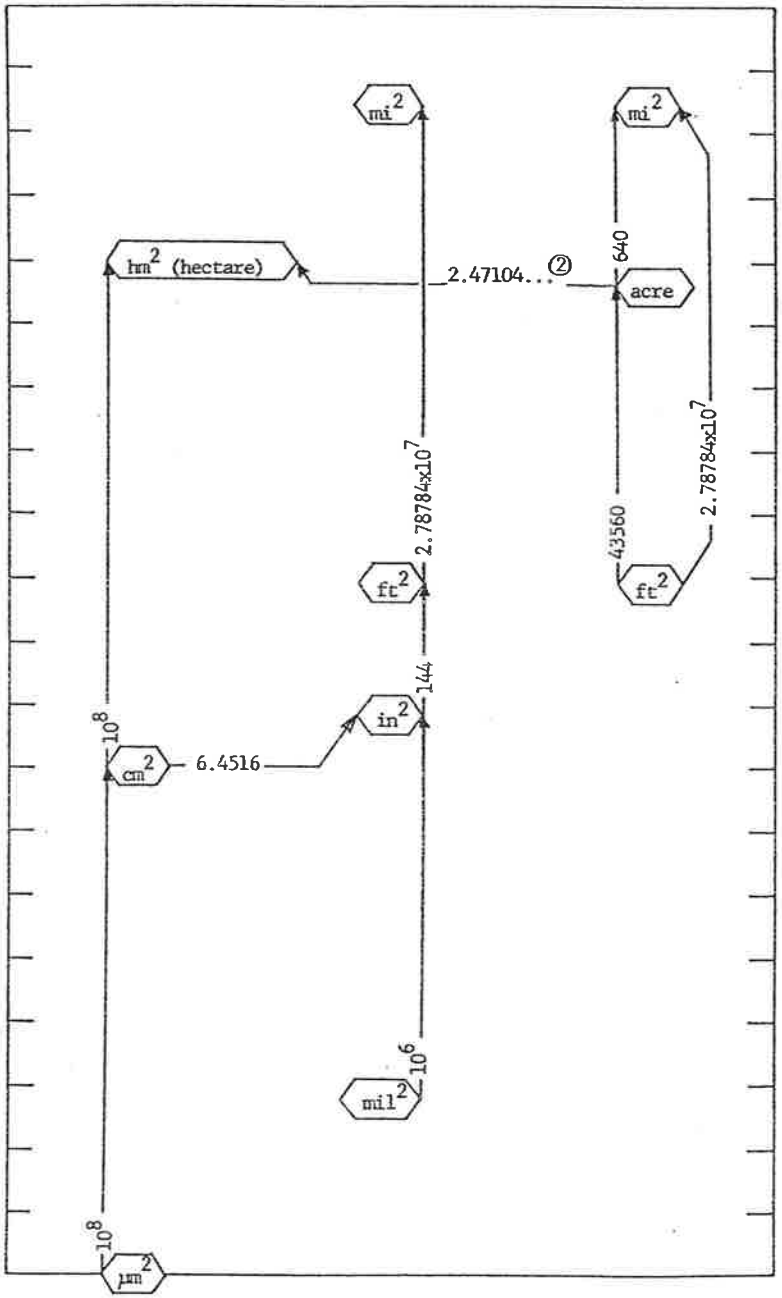


Figure C-3 Area Conversion Factors

Metric US Customary
International Survey



Appendix D SUMMARY OF CENTRAL SUBROUTINES AND PROGRAMS

Tables D-1 and D-2 list the properties of the central subroutines and programs from chapter 1 through chapter 8. The column labeled "User-supplied variables" lists the variables which a subroutine or program prompts the user for. The column labeled "Caller-supplied variables" lists the variables which are supplied by a calling main program or subroutine. The column labeled "Output variables" lists the variables which a central subroutine returns to the calling main program or subroutine. The column labeled "Variables changed" lists all variables changed by the central program or subroutine, as well as those changed by any subroutines which are called by that central subroutine or program.

Table D-1 Summary of Subroutines and Programs

Name:	Description:	User-Supplied Variables:	Caller-Supplied Variables:	Output Variables:
VINAX	Vector minimum and maximum	-	N X$	U V
VDOTP	Vector dot-product, etc.	-	N X$	V
POINT.1	Data-point-inputing subroutine	N X() Y()	-	N X() Y()
POINT.2	Function-sampling subroutine	A B N Y$	-	N X() Y()
SCALE.1	Axis endpoint-finding subroutine (Integer*10**X)	-	N X$	U V
SCALE.2	Axis endpoint-finding subroutine (Min, max)	-	N X$	U V
PLOTD	Data-point-plotting subroutine	-	N X() Y()	-
PLOT2	Plane-curve-plotting program	A B C D E F X$ Y$ Z	-	-
PLOT3	Contour-plotting program	A B C D M Z$	-	-
ERFCX	Complementary Error Function	-	X	V
GAMAX	Gamma Function	-	X	V
BESEX	Integer-order Bessel Functions of 1st Kind	-	N X	V
DERIV.1	General-order derivatives of sampled function	-	G X() Y()	V
INTEG.1	General Simpson's Rule integration	-	N X() Y()	Q R S T
INTEG.2	10-Point Gaussian integration program	A B Y$	-	-
MXINV	Matrix inverter with determinant	-	N X$	V(,)
MXMUT	Matrix multiplying subroutine	-	N L M X$	U(,)
ROOTN	Newton root-finder (Systems of equations)	F$() N X() X$	-	-
REGRP	General linear-least-squares regression	-	F$(),N,P,Y()	W()
REGRN	General non-linear-least-squares regression	-	F$,N,P	W()
RKUTT	4th-order Runge-Kutta (Systems of diff. eqns.)	-	A B E F$() H Y()	N V(,)
RATIO	Rational-fraction-finding program	E	-	-
OPERA	Multiple-precision operations (*,+,-)	-	N X$ Y$ Z(,)	Z(,)
BASER	Base-changing program (Integers and fractions)	A B X$	-	-
ZXCAL	ZX81 calculator	Y$	-	-

Table D-2 Summary of Subroutines and Programs

Name:	Starting Line no.:	Sub/Prog:	Other Subroutines Called:	Variables Changed:
VINAX	1000	SUB	-	I M U V
VDOTP	1100	SUB	-	I V
POINT.1	2000	SUB	-	I N X() Y()
POINT.2	2100	SUB	-	A B I N X() Y() Y$
SCALE.1	2200	SUB	VINAX	I M P U V
SCALE.2	2300	SUB	VINAX	I M U V
PLOTD	2400	SUB	SCALE.1 or .2	A B C D I M P U V X$
PLOT2	2500	PROG	-	A B C D E F M T X X$ Y Y$ Z
PLOT3	2600	PROG	-	A B C D E F M X Y Z Z$
ERFCX	3000	SUB	-	I V W()
GAMAX	3100	SUB	-	I M V W()
BESEX	3200	SUB	-	I M U V
DERIV.1	4000	SUB	-	D() J K V
INTEG.1	4200	SUB	DERIV.1	D() D1 D2 D3 D4 G I I1 I2 I3 J K Q R S T V
INTEG.2	4300	PROG	-	A B I S W() X() Y$
MXINV	5000	SUB	-	I J K L M U U(,) V(,)
MXMUT	5100	SUB	-	I J K U(,)
ROOTN	6000	PROG	MXINV MXMUT	A(,) D() F() F$() I J K L M N P U U(,) V(,)X() X$
REGRP	6100	SUB	MXINV MXMUT	G() I J K L M U U(,) V(,)W() X$ Z(,)
REGRN	6200	SUB	MXINV MXMUT	B() D() F G() I J K L M Q U U(,) V(,) W() X$ Z(,)
RKUTT	7000	SUB	-	I J K K(,) N V(,) X X$
RATIO	8000	PROG	-	A B C E
OPERA	8100	SUB	-	A A$ B C D E F G H I J X() Y() Z(,)
BASER	8600	PROG	-	A A() B B() I J K M N R S V X$
ZXCAL	8800	PROG	-	N X$ Y$

REFERENCES

1. Milton Abramowitz and Irene Stegun, eds., Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables, N.Y., Dover, 1972.

2. G. Arfken, Mathematical Methods for Physicists, N.Y., Academic Press, 1970.

3. Petr Beckmann, A History of ¶ (PI), N.Y., St. Martin's Press, 1971.

4. Richard Booth, "Curve-Plotting Graphics," SYNC Magazine, vol. 2, no. 6, pp. 29-35.

5. G. Forsythe, Malcom, and Moler, Computer Methods for Mathematical Computations, N.J., Prentice-Hall, 1977.

6. David Goldman and R. Bell, "The International System of Units (SI)," NBS Special Publication 330, Dec. 1981.

7. Michael Greenberg, Foundations of Applied Mathematics, N.J., Prentice-Hall, 1978.

8. Morris Hirsch and Stephen Smale, Differential Equations, Dynamical Systems, and Linear Algebra, N.Y., Academic Press, 1974.

9. John Heilborn, ed., Science and Engineering Programs, Apple II Edition, Berkeley, CA, Osborne/McGraw-Hill, 1981.

10. William Hayt, Jr., Engineering Electromagnetics, N.Y., McGraw-Hill, 1974.

11. Lawrence Huelsman, Basic Circuit Theory with Digital Computations, N.J., Prentice-Hall, 1972.

12. L. Jolley, Summation of Series, N.Y., Dover, 1961.

13. Donald Kreider, Kuller, Ostburg, and Perkins, An Introduction to Linear Analysis, Reading, MA, Addison-Wesley, 1966.

14. J. Dennis Lawrence, A Catalog of Sppecial Plane Curves, N.Y., Dover, 1972.

15. Allan Martin and Victor Mizel, Introduction to Linear Algebra, N.Y., McGraw-Hill, 1966.

16. John McNamee, "A Program to Integrate a Function Tabulated at Unequal Intervals," International Journal for Numerical Methods in Engineering, vol. 17, 1981, pp. 271-279.

17. "Factors for High-Precision Conversion ... U.S. Customary and Metric Units," NBS Letter Circular LC1071, July 1976.

18. Jon Passler, "Linear Regression," SYNC Magazine, vol. 2, no. 1, pg. 32.

19. Jon Passler, "DEF on the Sinclair," SYNC Magazine, vol. 2, no. 4, pp. 64-66.

20. L. Pipes and S. Hovanessian, Matrix Computer Methods in Engineering, Wiley, 1969.

21. Alger Salt, "Least Squares Data Analysis with the ZX80/81," SYNC Magazine, vol. 2, no. 3, pp. 32-36.

22. Barry Taylor, et al., "Particle Data Group: Review of Particle Properties," Reviews of Modern Physics, vol. 52, no. 2, Part II, April 1980, pg. 533.

23. George Thomas, Jr., Calculus and Analytic Geometry (Alternate Edition), Reading, MA, Addison-Wesley, 1972.

24. Hugh Young, Statistical Treatment of Experimental Data ... An Introduction to Statistical Methods, N.Y., McGraw-Hill, 1962.

25. Basil Wentworth, "Solving Implicit functions on the ZX81," SYNC Magazine, vol. 2, no. 5, pp. 58-61.

Please use the following page to send in any comments, suggestions interesting results, new programs, etc. If you find any glaring errors in the book, please correct me. If you have succeeded in translating any of these programs to different computers, please tell me how you did it, and what problems you had in doing it. If there is a second printing, I may include an extra chapter of programs sent in by readers.

RICHARD BOOTH
SHERMAN [illegible]
[illegible], PA 18015

VOID VOID VOID